FIRMWARE MODULES AND TABLES

C000-C6BF	**Math, utilities**
C6C0-C718	**Bank switching**
C719-D100	**BASIC handler**
CBBF-CD8A	Strings BASIC commands
CD8B-CF01	Pointers to strings BASIC commands for List
CF02-CF7D	Pointers to execution routines BASIC commands
CF86-CF90	Table prefixes unitary operations
CF91-CFD7	Table binary operators
CFD8-CFE5	Table unitary operators
CFE6-D100	Table strings BASIC functions + Data
D101-D194	**String handler**
D195-D23C	**Heap handler**
D23D-D8FA	**I/O hand**
D8FB-D9F4	**Interrupt handler**
D9F5-DAD3	**Error handler**
DA94-DACB	Pointers to strings error messages
DC1C-DD19	Strings error messages
DAD4-DDD0	**Print routines**
DB6F-DC1B	Strings machine messages
DDD1-DE01	**Encoding service routines**
DE02-DEB4	**Single/double byte utilities**
0DEB5-0ECAA	**BASIC execution/run-time module**
0E9F0-0EA3F	Function indirection table
0ECAB-0EFFF	**List handler**
0ECFB-0ED39	Pointers list handling routines
1E000-1EE6D	**Math package**
1EE6E-1EFFF	**Sound module**
2E000-2EBF3	**Screen driving package**
2E030-2E0C2	Screen constants
2EBF4-2EFFF	**Editor package**
3E000-3E9FF	**Encoding package**
3E8C5-3E934	ASCII tables
3EA00-3EFFF	**Utility package**

MEMORY MAP

0000-003F	INTERRUPT VECTOR ROUTINES
0047-006F	UTILITY WORK AREA
0072-00CF	SCREEN VARIABLES
00D0-00FF	MATH. WORKING AREA
0100-02EB	BASIC VARIABLES:
02EC-BFFF	HEAP, PROGRAM AREA, SCREEN RAM
C000-F8FF	ROM AND CPU AREA
F900-FFFF	I/O DEVICE ADDRESSES

PREFACE

Well, the work is done now. After 2 years of investigation and typing the DAI personal computer software manual is ready. A countless number of hours resulted in about 500 pages of Assembler listings.

Going through the DAI firmware will show you all the good (and less good) features of the DAI personal computer. Many routines can be used in your own machine language programs. Therefore, entry and exit conditions are added to the routines.

Please don't blame me if you may find some wrong interpretations of parts of the firmware. It is sometimes very difficult to trace the ideas of the one who wrotes the program.
If you may have any comments, I woud be very grateful if you could transmit these to me in writing. It may result in updates of the manual, published seperately or in the DAInamic Newletter.

I would like to acknowledge the DAInamic Users Club for the support with all the information they had available.
I would like to especially thank Mr. Gordon Wassermann for using the results of his investigations of the firmware.
The chapter 'Updates BASIC V1.1' is a result of the compare Jos Schepens did on both BASIC versions.

Jan Boerrigter - September, 1982.

Published by:
'Micro Service'
Fabritiusstraat 15,
6174 RG Sweikhuizen
The Netherlands.

SUMMARY:
========

C000	C6BF:	Math. utilities.
C6C0	C718:	Bank switching.
C719	D100:	BASIC handler.
D101	D194:	String handler.
D195	D23C:	Heap handler.
D23D	D8FA:	I/O handler.
D8FB	D9F4:	Interrupt handler.
D9F5	DAD3:	Error handler.
DAD4	DDD0:	Print routines.
DDD1	DE01:	Encoding service routines.
DE02	DEB4:	Single/double byte utilities.
DEB5	0ECAA:	BASIC execution/run-time module.
0ECAB	0EFFF:	List handler.
1E000	1EE6D:	Math. package.
1EE6E	1EFFF:	Sound module.
2E000	2EBF3:	Screen driving package.
2EBF4	2EFFF:	Editor package.
3E000	3E9FF:	Encoding package.
3EA00	3EFFF:	Utility package.

'Gaps' in these modules are filled with parts of routines from other modules.

TABLES:
=======

CBBF:	Strings BASIC commands.
CD8B:	Pointers to strings BASIC commands (for LIST).
CF02:	Pointers to execution routines BASIC commands.
CF86:	Table prefixes unitary operations.
CF91:	Table binary operators.
CFD8:	Table unitary operators.
CFE6:	Table strings BASIC functions.
DA94:	Pointers to strings error messages.
DB6F:	Strings machine messages.
DC1C:	Strings error messages.
0E9F0:	Function indirection table.
0ECFB:	Pointers LIST handling routines.
2E030:	Screen constants.
3E8C5:	ASCII tables.

DAI pc MEMORY MAP

BASIC V1.0/V1.1 Revision 5.1 Date: 18.9.82

INTERRUPT VECTOR ROUTINES: 0000 - 003F

```
0000-07          Interrupt vector routine 0:
                    Used by Utility (LOOK).
0008-0F          Interrupt vector routine 1:
                    Used by Utility and encoding Basic.
                    RST 1 + data: Switch to ROM-bank 3.
0010-17          Interrupt vector routine 2:
                    Used by stack interrupt.
0018-1F          Interrupt vector routine 3:
                    Used by sound interrupt.
0020-27          Interrupt vector routine 4:
                    Used for math. routines.
                    RST 4 + data: Switch to ROM-bank 1.
0028-2F          Interrupt vector routine 5:
                    Used for screen handling routines.
                    RST 5 + data: Switch to ROM-bank 2.
0030-37          Interrupt vector routine 6:
                    Used for keyboard service routines.
0038-3F          Interrupt vector routine 7:
                    Used to flash the cursor.

Interrupt vector    00  NOP
routines:           E5  PUSH H
                    2A  LHLD:
                    ..  ) vector address location
                    ..  ) see (#0062-#0071).
                    E9  PCHL
                    00  NOP
                    00  NOP
```

BANK SWITCHING AREA: 0040 - 0046

```
0040     !POROM:  ) Memory of last outputs to output ports.
         !POR1M:  ) Duplicate of (#FD06).
          POROM:  )
0041/42  RSWK1:   Save PSW during ROM bank switching.
0043/44  RSWK2:   Save HL during ROM bank switching.
0045/46           Spare.
```

UTILITY WORK AREA: 0047 - 0061

```
0047                Store EI/DI instructions after using LOOK
                    the first time (No clear occurs).
0048/49             High address trace window.
004A/4B             Low address trace window.
004C-4F             Store current instruction if LOOK is used,
                    preceeded by EI. In case of a RST-instruction is
                    stored: RSTx/data x; RST0. In case of an EI instruction
                    is stored: EI, NOP, next instruction.
0050                Flag for Look initialisation:
                       #FF: init. Look, else: #00.
0051/52  IADR:      I: Address current instruction.
0053     AFSAV:     A: Contents A after execution of I.
0054                F: Idem status flags.
0055     BCSAV:     B: Idem B register.
0056                C: Idem C register.
0057     DESAV:     D: Idem D register.
0058                E: Idem E register.
0059     HLSAV:     H: Idem H register.
005A                L: Idem L register.
005B/5C  SPSAV:     S: Idem stackpointer.
005D/5E  PCSAV:     P: Address next instruction to be
                       executed.
005F     TICIM:     M: Current interrupt mask.
                       Duplicate of (#FFF8)
0060                T: Value TICC control word.
                       (#FC after Z2).
0061                G: Value GIC control word.
                       (#1B after Z2).
```

INTERRUPT VECTOR ADDRESSES: 0062 - 0071

```
0062/63  I0USA:   Vector address RST 0: set by UT (Z2): 3#EB5D.
0064/65  I1USA:   Vector address RST 1: utility/encode:  #C70E.
0066/67  I2USA:   Vector address RST 2: stack interrupt: #D9E2.
0068/69  I3USA:   Vector address RST 3: sound interrupt: #D755.
006A/6B  I4USA:   Vector address RST 4: math. restart:   #C6C0.
006C/6D  I5USA:   Vector address RST 5: screen restart:  #C6FD.
006E/6F  I6USA:   Vector address RST 6: keyb. int. serv: #D578.
0070/71  I7USA:   Vector address RST 7: clock interrupt: #D9A9.
                  By changing the vector addresses, other
                  interrupt routines can be used.
```

SCREEN VARIABLES: 0072 - 00CF

Character mode variables:

```
0072/73  CURSOR:  Cursor position address.
0074     CURTY:   Cursor type:
                     #00: cursor flashes in colour.
                     #01: cursor alternates between actual
                          character and contents #0075.
```

```
0075      CURIN:  Cursor information:
                     If type = 0: Mask which is EXOR'ed with
                                  the colour byte for that
                                  character to flash it.
                     If type = 1: Cursor alterates between
                                  actual character and this
                                  information.
0076/77   CURSV:  Contents screen RAM location indicated by
                  the cursor:
                     #0076 contains the colour byte,
                     #0077 contains the data.
0078/79   LNSTR:  Address line mode byte of currently used
                  line of the screen RAM.
007A      LNEND:  Lobyte of end of cursor line.
                  Used to check if end of line is reached.
007B      LCONT:  Number of extended lines.
007C      COLMT:  ) colours for colour    #80+X3
007D              ) registers COLORT      #90+X2
007E              )                       #A0+X1
007F              )                       #B0+X0

          Variables set to describe the current state of
          the screen:

0080/81   SCREEN: Points to first byte of screen RAM (#BFFF).
0082/83   SCTOP:  Points after header (#BFEF).
0084/85   FFB:    First free byte in this mode.
0086/87   GRR:    Points to top of rolled area. Contains the
                  line mode byte of the line where split mode starts.
0088/89   GRE:    Points after end of graphics area.
008A/8B   CHS:    Points to start of character area.
008C/8D   GAE:    Unsplit: End archive area.
          CHE:    Split: After end of character area.
008E/8F   SCE:    End of screen (after trailer).
0090/91   GTE:    End area used splitting mode.
0092/93   GAS:    Unsplit: start archive area.
          GTS:    Split: start temporary save area.
0094/95   GRC:    Number of blobs horizontally in mode.
0096      GRL:    Number of lines of graphics in mode.
0097      GAL:    Number saved lines of graphics.
0098      GXB:    Number of bytes/line this mode.
0099/9A   GREQ:   Previous end of graphics.
009B/9C   CHSO:   Previous start characters:
                  Was split: previous mode byte of 1st text line.
                  Was graphics: Previous last COLORT-byte.
009D      SMODE:  Current screen mode (updated after mode
                  changed):
                     #00 mode 1          #08 mode 5
                     #01 mode 1A         #09 mode 5A
                     #02 mode 2          #0A mode 6
                     #03 mode 2A         #0B mode 6A
                     #04 mode 3
                     #05 mode 3A         #10 during init.
                     #06 mode 4
                     #07 mode 4A         #FF mode 0
                        bits 4-7 are ignored;
                        #0C,0D,0E,0F are inhibited.
```

```
          Graphics mode variables (From #00A2-#00B5 also
          used by the EDIT mode):

009E      COLMG:   ) colours for colour #80+X3
009F               ) registers COLORG   #90+X2
00A0               )                    #A0+X1
00A1               )                    #B0+X0
00A2      SCVR:
00A3-AA   SCXBUF:  Buffer used to hold contents of an 8 bit
                   field during 16 colour updates.
00AB      SBGOU:   Flags if colour is being carried out to
                   next field.
00AC      SBGOC:   Colour being carried out.
00AD-B4   COLS:    Buffer for impossible requests.

         !Edit variables:

00A2/A3 !EBUFR:   Address start EDIT buffer.
00A4/A5 !EBUFN:   Address end of text in EDIT buffer.
00A6/A7 !EBUFS:   End available space in EDIT buffer.
00A8    !EWINX:   Offset of left side of window.
00A9/AA !EWINY:   Offset of top of window from start buffer.
00AB    !ECURX:   X-offset of cursor in document (current
                  cursor position in text line).
00AC/AD !ECURY:   Y-offset of cursor in document (count of
                  current cursor line).
00AE/AF !CURPT:   Pointer to cursor position in buffer.
00B0/B1 !CURLS:   Pointer to line mode byte of cursor line on screen.
00B2/B3 !CURLB:   Pointer to start of cursor line in buffer.
00B4/B5 !TABTP:   Address tab position table.

          Line drawing variables:

00B5/B6   DELTA:  Amount to add into count.
00B7/B8   RT:     Count.
00B9/BA   COR:    Adjustments for long sectors.
00BB/BC   SECT:   Lower of 2 possible sector lengths.
00BD      SECTC:  Number of sectors.
00BE      TRIM:   Amount to trim off last sector.
00BF      DIRN1:  Set if Y-direction is negative.
00C0      DIRN2:  Set if swap X,Y directions.
00C1      ANIM:   Set if animate in 4 colour mode.
00C2/C3   FCOLR:  Details of colour required.


00C4/C5   ASMKRM: Address memory management routine (#CA01).
                  Checks available RAM space.
00C6/C7   AESTOP: Address emergency stop routine (#CA25).
                  Return-routine for 'Out of space for mode'.
00C8-CF           Spare.


MATH. WORKING AREA: 00D0 - 00FF
===============================

00D0/D1   EVECT:  Pointer to table with error routines (#C7F2).
00D2/D3   AGETC:  Pointer to input routine (#DDE0).
00D4      MVECA:  Math. chip flag: offset of start HW/SW vector:
                  (offset for RST 4 restart routines):
                     #00  No math. chip.
                     #7B  math. chip present.
```

```
00D5-D8  FPAC:    ) Arithmetic FPT/INT accumulator.
         IAC:     )
00D9     SF:      Subtraction flag.
00DA     OP4:     Operand 4th byte.
00DB     OP3:     Operand 3rd byte.
00DC     OP2:     Operand 2nd byte.
00DD     OP1:     Operand 1st byte.
00DE     EXPDF:   Difference in exponents for last FPT
                  add/sub operations.

         Work area for math. operations:

00DF/E0  FWORK:   ) Also used for data save during stack operations.
         XPRAS:   )
00E1/E2  XPHLS:   )

         FPOLY variables (RAM shared with SQRT):

00E3-E6  XN:      Running power of X (X^K).
00E7-EA  XK:      Power multiplier (X^J).
00EB-EE  SUM:     Running sum.

         SQRT variables (RAM shared with FPOLY):

00E3-E6  F:       Mantissa.
00E7-EA  P:       Polynomial approximation.

         EXP variables (RAM shared with TRIG, INVTRIG):

00EF     SIGN:    Input sign.

         TRIG variables (RAM shared with EXP, INVTRIG):

00EF-F2  FTWRK:   Work location for TAN.

         Inverse TRIG variables (RAM shared with EXP, TRIG):

00EF-F2  FATZX:   Z,X. Used by ATAN, ASIN, ACOS.

         Number input variables:

00E3-E6  ICBWK:   Number to add for each digit.

         Number output variables:

00E3-F1: DECBUF:  Decimal output buffer.
                     MAXSIG: #0A: Max. possible significant figures.
                     FPTSIG: #06: Number of significant digits for FPT.
00E4     DECBS:   Sign.
00E5     DECBD:   Decimal point.
00E6-F0  DECBF:   Digits, most significant one in 00E6.
00F1     DECBE:   Exponent.
00F2/F3  DECBP:   Buffer pointer.
00F4-FF           Spare variable space.
```

BASIC VARIABLES: 0100 - 02EB

User state:

Following are saved by soft break: (SFRAME = SYSTOP - SYSBOT)

```
0100/01  SYSBOT: ) Start of current line. Points to first
         CURRNT: ) byte of line number.
0102/03  BRKPT:  Start of current command.
0104/05  LOPVAR: Points to current loop variable. Points to
                 position of variable in symbol table.
                    #00 if no running loop.
0106     LSTPF:  Flag for integer/fpt loop and
                 implicit/explicit step.
                    bit 0: 0 = implicit step.
                           1 = expicit step.
                    bit 7: 0 = FPT loop variable.
                           1 = INT loop variable.
0107-0A  LSTEP:  Step value if explicit.
010B-0E  LCOUNT: Loop iteration count.
010F-10  LOPPT:  Pointer to start address loop.
0111/12  LOPLN:  Pointer to start loop line.
0113/14  STKGOS: Stack level at last GOSUB.
                    #00 if no active call.
0115     SYSTOP: )
        (STRFL:  ) Trace/step flag together)
         TRAFL:  ) Trace flag (#FF if set).
0116     STEPF:  Step flag (#FF if set).
0117     RDIPF:  Flag set while running input (set: #FF).
0118     RUNF:   Flag set while running program.
                 (Previous 2 bytes must be consecutive)
```

Runtime scratch area:

```
0119/1A  GSNWK:  Scratch area for GOSUB/NEXT (2 bytes).
                 Points to destination address last GOSUB.
         LISW1:  Startaddress of listed area.
         COLWK:  Scratch area for SCOLG, SCOLT (4 bytes).
                 Contains last selected COLORT/COLORG values.
011B/1C  LISW2:  End address listed area.
```

Save area for restart on error:

```
011D/1E  ERSSP:  Stack pointer.
011F-21
0122     ERSFL:  Set if encoding a stored line (set: #01).
```

Data/read variables:

```
0123     DATAC:  Offset of next character to encode.
0124/25  DATAP:  Pointer to address current data line.
        !DATAQ:  Pointer after current data line.
0126     CONFL:  Set if there is a suspended program (set: #01).
0127/28  STACK:  Current base stack level.
```

Scratch location for expression/function evaluation.

```
0129-2C  WORKE:  Scratch area. Contains also the argument A of
                 the last software random RND(A).
```

Random number kernel:

012D-30 RNUM: Random number kernel.
!RNDLY: Random number delay count (1 byte).

Output switching:

0131 OTSW: #00 output to screen + RS232.
#01 output to screen only.
#02 output to edit buffer.
#03 output via DOUTC.

Encoding input source switching:

0132/33 EFEPT: Encoding input pointer. Points to start-address of Basic-line just being encoded.
0134 EFECT: Encoded input count. Counts length of line.
0135 EFSW: Encoded input switching:
#00 Input from keyboard/DINC.
#01 Input from string.
#02 From edit buffer to program area.

Variables used during expression encoding (could overlap with runtime variables):

0136 TYPE: Type of latest expression or item:
#00 FPT
#10 INT
#20 STR
#30 Boolean
0137 RGTOP: Latest priority operator:

#00	no operation	#6A	IOR
#38	AND	#6C	IXOR
#39	OR	#8D	SHL
#50	>=	#8E	SHR
#51	>	#A0	+
#52	<>	#A1	-
#53	<=	#C2	/
#54	<	#C3	*
#55	=	#CF	MOD
#69	IAND	#E4	^

013B OLDOP: Old priority operator.
0139/3A HOPPT: Pointer to place in encoded input buffer for next operator.
013B/3C RGTPT: Pointer to place in encoded input buffer of operand latest operator.

Mask to select cassette 1 or 2:

013D CASSL: #10 Cassette 1 activated.
#20 Cassette 2 activated.

Encoded input buffer:

013E-BD EBUF: 128 bytes buffer. Also used by utility.

Interrupt handler variables:

01BE/BF TIMER: Timer location. Also used in WAIT TIME.
01C0 CTIMR: Cursor clock. Used for cursor flashing.
CTIMV: #0F: Flash time in 20 ms units.
If #00, cursor flashes.
01C1 KBXCT: Extend keyboard scan time counter. When #00, keyboard scan will be performed.
KBXCK: #02: Keyboard scan time (16 ms units). Also used by RAND routine.

Sound control block storage:

01C2-CF: Sound control block 0.
SCBL: Length of a sound block (14 bytes).
01C2 SCB0: Elapsed count of current volume:
#FF: channel off.
#FE: current volume forever.
01C3/C4 Pointer to required count at this volume in envelope table.
01C5/C6 Pointer to start envelope table being used.
01C7 Sound-volume *8. Multiplier for volume, between 0 and 16, shifted 3 places left.
01C8 Basic volume at this moment, calculated from sound-volume and present envelope volume.
01C9 Counter for tremolo. 0 if no tremolo.
01CA Actual volume, calculated from volume and tremolo fluctuations.
01CB Glissando flag:
#00 Endperiod reached.
#01 Set frequency.
#02 Endperiod not reached.
01CC/CD Current period of output.
01CE/CF Required final period of output.
01D0-DB SCB1: Sound control block 1 (see SCB0).
01DE-EB SCB2: Sound control block 2 (idem).
01EC-F4 NCB: Noise control block.
NCBL: Length of noise block (9 bytes).
The noise control block is identical to the sound control block, but without period-values and tremolo.

Envelope storage:

01F5--0274 ENVST: Envelope storage (128 bytes).
ENVLL: #40: Number of bytes/envelope
NUMENV: #02: Number of envelopes.
Two envelope tables of each 64 bytes:
#01F5-#0234 and #0235-#0274.

Type storage:

0275--28E IMPTAB: Implicit type table.

#0275	A	#027C	H	#0283	O	#028A	V
#0276	B	#027D	I	#0284	P	#028B	W
#0277	C	#027E	J	#0285	Q	#028C	X
#0278	D	#027F	K	#0286	R	#028D	Y
#0279	E	#0280	L	#0287	S	#028E	Z
#027A	F	#0281	M	#0288	T		
#027B	G	#0282	N	#0289	U		

```
028F      IMPTYP: Default number type. Selected by IMP command.
                  #00  FPT
                  #10  INT
                  #20  STR
0290      REQTYP: Required number type for present operation.
                  #00  FPT
                  #10  INT
                  #20  STR
                  #30  Variable name argument
                  #40  Array without arguments


0291/92   DATAQ:  Pointer to begin current data line.
0293      RNDLY:
0294      POROM:  Duplicate of (#FD04).
0295      POR1M:  Duplicate of (#FD05).
0296      INSW:   Input switching:
                     If #00, input from keyboard.
                     If <>#00, input from DINC (Default: RS232).
0297-9A           Spare.

          Heap/text buffer/symbol table pointers:

029B/9C   HEAP:   Start address of HEAP.
029D/9E   HSIZE:  Size of HEAP.
                     HSIZD: #100: Default size.
029F/A0   TXTBGN: Start address of text buffer.
02A1/A2   TXTUSE: End text buffer and.
          STBBGN: Start symbol table.
02A3/A4   STBUSE: End of symbol table.
02A5/A6   SCRBOT: Bottom screen RAM area (48K):
                     mode 0:       #B350
                     mode 1/2(A):  #B7A0
                     mode 3/4(A):  #A65C
                     mode 5/6(A):  #63B8

          Keyboard variables + constants:

02A7/A8   KBTPT:  Pointer to table with ASCII-codes.
02A9-B0   MAP1:   Latest scan of keys (key-codes).
                  (row 0 in #02A9, row 7 in #02B0)
   02AF   RPLOC:  Byte containing REPT key.
                     RPMSK: #20: Rept key bit.
                     BRSEL: #40: Column select mask for BREAK.
                     BRMSK: #40: Break key bit.
   02B0   SHLOC:  Byte containing SHIFT.
                     SHMSK: #40: Shift key bit.
02B1-B8   MAP2:   Previous scanning of keyboard.
02B9      KNSCAN: Set to scan for BREAK only. When (#02B9)
                  is #FF: scan for BREAK only.
02BA-BD   KLIND:  4 byte circular buffer to store the ASCII
                  values for keys pressed.
                     KBLEN/KEYL: #04: length rollover buffer.
02BE/BF   KLIIN:  Next position for input to KLIND.
02C0/C1   KLIOU:  Next position for output from KLIND.
02C2      RPCNT:  Count for REPT. #01 if REPT is not
                  pressed. Else it is used as timer for the
                  repeat function.
02C3      SHLK:   Set to #FF if CTRL is pressed to
                  invert SHIFT. Else #00. Used to
                  calculate the offset for the ASCII code
                  table.
```

```
02C4      KBRFL:  Break flag. #FF indicates BREAK pressed
                  (Only if suspended program). If BREAK is pressed,
                  #02C4 counts from 00 to #0F before stopping
                  the program.

          Data/cassette switching vectors:

          Copy of ROM (#D7A4 - #D7CA) for cassette and RS232.
          Can be loaded with other I/O vectors.

02C5-EB   IOVEC:  02C5  WOPEN:  C3 B8 D2  JMP: D2B8
                  02C8  WBLK:   C3 F1 D2  JMP: D2F1
                  02CB  WCLOSE: C3 27 D4  JMP: D427
                  02CE  ROPEN:  C3 25 D3  JMP: D325
                  02D1  RBLK:   C3 40 D3  JMP: D340
                  02D4  RCLOSE: C3 45 D4  JMP: D445
                  02D7  MBLK:   C3 A2 D3  JMP: D3A2
                  02DA  RESET:  C9 00 00  RET
                  02DD  DOUTC:  C9 00 00  RET
                  02E0  DINC:   C3 B4 DD  JMP: DDB4
                  02E3          C9 00 00  RET
                  02E6  TAPSL:  24 24     Tape speed leader.
                  02E8  TAPSD:  24 3C     Tape speed data.
                  02EA  TAPST:  24 18     Tape speed trailer.


HEAP, PROGRAM AREA, SCREEN RAM: 02EC - BFFF
===========================================

02EC-   (RAM:      HEAP (Strings + arrays)   - See (#029B/9C).
 -BFFF  (VAREND:   Program (compiled Basic)  - See (#029F/A0).
        (VARLAST:  Symbol table              - See (#02A1/A2).
                   Not used RAM              - See (#02A3/A4).
                   Screen RAM                - See (#02A5/A6).


ROM AND CPU AREA: C000 - F8FF
=============================

C000-             24K ROM:
  -EFFF              #C000-#DFFF: 8K non-switched ROM.
         VECA:       #E000-#EFFF: 4 banks of each 4K ROM.
                                  (switchable).


F000-             Can be used for ROM extension (reading only).
  -F7FF           Is already completely used by Memocom MDCR-D.


F800-             Microcomputer stack.
  -F8FF           Incl. vector for MDS jump instructions.
                     #F800  SRBOT    Bottom of stack RAM.
                     #F900  STTOP    Top of stack RAM.
```

I/O DEVICE ADDRESSES: F900 - FFFF

F900-
-FAFF Spare I/O device addresses.
(Not wired on pC board).

MATH. CHIP AMD 9511: FB00 - FBFF

FB00 MTHAD:) Data math.chip.
MDATA:)
FB02 MCOMD:) Command + status.
MSTATUS:)

AMD9511 operator and status bytes:

ODADD:	#2C Int addition	OFADD:	#10 Fpt addition
ODSUB:	#2D Int subtract	OFSUB:	#11 Fpt subtract
ODMUL:	#2E Int multiply	OFMUL:	#12 Fpt multiply
ODDIV:	#2F Int division	OFDIV:	#13 Fpt division
OSQRT:	#01 Square root	OFIXD:	#1E Fix
OSIN:	#02 Sine	OFLTD:	#1C Float
OCOS:	#03 Cosine	OCHSD:	#34 Change sign int
OTAN:	#04 Tangent	OCHSF:	#15 Change sign fpt
OASIN:	#05 Arc sine	OPTOD:	#37 Push int/fpt
OACOS:	#06 Arc cosine	OPOPD:	#38 Pop int/fpt
OATAN:	#07 Arc tangent		
OLOG:	#08 Log base 10		
OLN:	#09 Log base e	MBUSY:	#80 Busy status bit
OEXP:	#0A Expotential	MERRB:	#1E All error bits
OPWR:	#0B X^Y	MZERO:	#20 Top of stack

PROGRAMMABLE INTERVAL TIMER 8253: FC00 - FCFF

Used for sound generator. 3 independent 16 bits
down counters with programmable counter modes.

FC00/01 SNDAD:)
SND0:) Counter 0 (oscillator channel 0).
!PDLCH: Used as counter for paddle operations.
FC02/03 SND1: Counter 1 (oscillator channel 1).
FC04/05 SND2: Counter 2 (oscillator channel 2).
(16 bit data; LSB first)
FC06 SNDC: Command 8253. To be loaded prior to freq.
selection with resp. #36, #76 and #B6.
Command word format:
bit 0 : 0 binairy counter 16 digits.
1 BCD counter (4 decades).
3,2,1: 000 mode 0: Interrupt on end count.
001 mode 1: Programmable one shot.
x10 mode 2: Rate generator.
x11 mode 3: Sq.wave rate generator.
100 mode 4: SW trig. strobe.
101 mode 5: HW trig. strobe.

```
                5,4  : 00  Counter latch operation.
                       01  Read/load MSB only.
                       10  Read/load LSB only.
                       11  Read/load LSB first, then
                           MSB.
                7,6  : 00  Select counter 0.
                       01  Select counter 1.
                       10  Select counter 2.
                       11  Illegal.
          Several control words:
             COFIX: #00  Fix count on channel 0.
             COM0:  #30  Chan.0, mode 0, 2 byte op.
             COM1:  #32  Chan.0, mode 1, 2 byte op.
             COM3:  #36  Chan.0, mode 3, 2 byte op.
             C1M3:  #76  Chan.1, mode 3, 2 byte op.
             C2M3:  #B6  Chan.2, mode 3, 2 byte op.


     DISCRETE I/O DEVICE ADDRESSES: FD00 - FDFF
     ==========================================

FD00   PORI:   IN (1)  bit 0: -
                       1: -
                       2: PIPGE: Page signal
                       3: PIDTR: Serial output ready
                       4: PIBU1: Button on paddle 1
                                 (1 = closed)
                       5: PIBU2: Button on paddle 2
                                 (1 = closed)
                       6: PIRPI: Random data
                       7: PICAI: Cassette input data

FD01   PDLST:  OUT (3) Single pulse used to trigger
                       paddle timer circuit.

FD04   PORO:   OUT (2) bit 0 - 3: volume osc. channel 0
                       4 - 7: volume osc. channel 1
FD05   POR1:   OUT (2) bit 0 - 3: volume osc. channel 2
                       4 - 7: volume random noise
                              generator.
FD06   PORO:   OUT (3) bit 0: POCAS: Cassette data output
                         1,2: PDLMSK: Paddle select
                           3: POPNA:  Paddle enable
                           4: POCM1:  Cassette 1 motor
                                      control.(0 = run)
                           5: POCM2:  Cassette 2 motor
                                      control.(0 = run)
                         7,6:         ROM bank switching:
                                         00  bank 0
                                         01  bank 1
                                         10  bank 2
                                         11  bank 3
```

PROGR. PERIPHERAL INTERFACE 8255 : FE00 - FEFF

Used for DCE-bus (GIC Controller).

FE00 GIC: (1) I/O port A
FE01 (1) I/O port B
FE02 (1) I/O port C
FE03 (6) Command word 8255:

Contr.	PA	PCH	PCL	PB	(mode 0)
#80	out	out	out	out	RWMOP
#81	out	out	in	out	
#82	out	out	out	in	
#83	out	out	in	in	
#88	out	in	out	out	
#89	out	in	in	out	
#8A	out	in	out	in	
#8B	out	in	in	in	
#90	in	out	out	out	RWMIP
#91	in	out	in	out	
#92	in	out	out	in	
#93	in	out	in	in	
#98	in	in	out	out	
#99	in	in	in	out	
#9A	in	in	out	in	
#9B	in	in	in	in	

TICC: TIMER + INTERRUPT CONTROLLER 5501: FF00-FFFF

FFF0 (4) Serial input buffer. Contains the last character received on the RS232 interface.

FFF1 (4) Keyboard input port. Bottom 7 bits are data input from the keyboard. Bit 7 is the IN7 line from the DCE-bus and is attached to the page-blanking signal for the TV. Every 20 ms. an impulse is present.

FFF2 (2) Interrupt address register:
bits 5,4,3: Number of pending interrupt.
7,6 :)
2,1,0:) always '1'

FFF3 (4) Status register:
bit 0: Frame error. Set by a BREAK on the RS232 input.
1: Overrun error. Set if a character has been received but not taken by the CPU.
2: Serial input. Set if no data is received.
3: Receive buffer loaded. Set if a character has been received.
4: Transmit buffer empty. Set if RS232 output is ready to accept another character.
5: Interrupt pending. Set if one or more of the enabled interrupts has occured.

```
                    6: Full bit detected. Set if the first
                       data bit of an incoming character
                       has been detected.
                    7: Start bit detected. Set if the start
                       bit of an incoming character has been
                       detected.
FFF4      (2) Command register:
              bit 0: TICC reset.
                  1: Send Break. If set, the serial output
                     is high impedance.
                  2: Interrupt 7 select. A '1' selects IN7
                     of the DCE-bus, a '0' selects Timer 5.
                  3: Interrupt acknowledge enable.
                     A '1' enables TICC to accept a INTA
                     signal from the CPU.
                  4 - 7: Always 0.
FFF5      (6) Communications rate register:
              bit 0:  110 baud
                  1:  150 baud
                  2:  300 baud
                  3: 1200 baud
                  4: 2400 baud
                  5: 4800 baud
                  6: 9600 baud
                  7: 1 - one stop bit
                     0 - two stop bits
FFF6      (6) Serial output buffer. Write byte to this
              location to send it on the RS232 output.
              Use only when #FFF3-bit 4 is high.
FFF7      (7) Keyboard output port. Data output to scan
              keyboard.
FFF8      (2) Interrupt mask register:
              bit 0: timer 1 has expired (UTIM).
                  1: timer 2 has expired.
                  2: External interrupt (STKIM).
                  3: Timer 3 has expired (SNDIM).
                  4: Serial receiver loaded.
                  5: Serial transmitter empty.
                  6: Timer 4 has expired (KBIM).
                  7: Timer 5 has expired or IN7 (CLKIM).
              (react only on low-high transition)
FFF9      (2) UTIAD:  Timer 1 address (UT).
FFFA      (1)         Timer 2 address.
FFFB      (2) SNDIAD: Timer 3 address (sound).
FFFC      (2) KBIAD:  Timer 4 address (keyboard).
FFFD      (1)         Timer 5 address.
FFFE          not used.
FFFF          not used.
```

NOTES:
(1) Read and write allowed.
(2) Reading allowed. Writing too, but may be overwritten by BASIC program.
(3) No writing allowed.
(4) Reading allowed, writing not.
(5) Should not be accessed.
(6) Writing allowed, reading not.
(7) Reading not allowed, writing is harmless but useless; keyboard scanner will overwrite it.

REMARKS:

ADDRESSES FB00 - FFFF:

The highest byte of the address is used for the chip select signal CS of the peripheral equipment 8253, 8255, 5501 etc. The lowest byte is used to address the several registers of the peripheral. Its high nibble does not have any value. So addresses in this range can be read as FBxO - FFxF, in which x is a don't care.

```
002                             ORG   :C000
003                     *
004                     *
005                     *
006                     *  ================
007                     *** MATH. UTILITIES ***
008                     *  ================
009                     *
010                     *
011                     ***************
012                     * ENTRYPOINTS *
013                     ***************
014                     *
015 C000 C319C7         BASE    JMP   :C719     Reset (entry on hardware
016                                               reset)
017 C003 C335C0         XINIT   JMP   :C035     Math. package initialisation
018 C006 C3F3C0         XFINM   JMP   :C0F3     Incr. FPT number in memory
019 C009 C3FBC1         XFDCM   JMP   :C1FB     Decr. FPT number in memory
020                                               (not used)
021 C00C C379C0         XFCOMP  JMP   :C079     FPT Compare
022 C00F C3BBC0         XIINM   JMP   :C0BB     Incr. INT number in memory
023 C012 C3D5C0         XIDCM   JMP   :C0D5     Decr. INT number in memory
024                                               (not used)
025 C015 C3ACC0         XICOMP  JMP   :C0AC     INT Compare
026 C018 C31EC2         XPUSH   JMP   :C21E     Save MACC on stack
027 C01B C334C2         XPOP    JMP   :C234     Retrieve MACC from stack
028 C01E C349C2         XFCB    JMP   :C249     Input FPT number to MACC
029 C021 C361C3         XFBC    JMP   :C361     Conv. FPT number for output
030 C024 C373C5         XICB    JMP   :C573     Input INT number to MACC
031 C027 C3B2C5         XIBC    JMP   :C5B2     Conv. INT number for output
032 C02A C314C6         XHCB    JMP   :C614     Input Hex number to MACC
033 C02D C353C6         XHBC    JMP   :C653     Conv. MACC to Hex for output
034 C030 C386C4         XPRTY   JMP   :C486     Pretties up FPT/INT number
035 C033 E300           DECBUF  DBL   :00E3     Location output buffer
036                     *
037                     ***********************************
038                     * MATH. PACKAGE INITIALISATION *
039                     ***********************************
040                     *
041                     * Entry: HL: Address input encoding routine (DDE0).
042                     *         DE: Base address error routines (C7F2).
043                     * Exit:  AFDEHL corrupted, BC preserved.
044                     *
045 C035 22D200         MINIT   SHLD  :00D2     Init. (00D2/3)=DDE0
046 C038 EB                    XCHG
047 C039 22D000                 SHLD  :00D0     Init. (00D0/1)=C7F2
048 C03C 3A02FB                 LDA   :FB02     Get math.chip status
049 C03F B7                     ORA   A         Check if math.chip present
050 C040 3E00                   MVI   A,:00     Flag = 0 if not
051 C042 FA47C0                 JM    :C047
052 C045 3E7B                   MVI   A,:7B     Flag = #7B if present
053 C047 32D400         LC18    STA   :00D4     Set math.chip flag
054 C04A C9                     RET
055                     *
056                     ***************************
057                     * OVERFLOW ERROR ROUTINE *
058                     ***************************
059                     *
060                     * (From LC20 common part for various entries).
061                     *
062                     * Jump to (00D0/1) = Address 'overflow error'
063                     * routine (C7F2).
```

```
064                     *
065                     * Entry: If start at LC20: offset in HL.
066                     * Exit:  AFBCDEHL preserved.
067                     *        On stack original returnaddress.
068                     *
069 C04B E5             FPEOV   PUSH  H
070 C04C 210000                 LXI   H,:0000   Init offset = 0
071                     *
072 C04F F5             LC20    PUSH  PSW
073 C050 D5                     PUSH  D
074 C051 EB                     XCHG            Offset in DE
075 C052 2AD000                 LHLD  :00D0     Get addr pointer
076 C055 19                     DAD   D         Add offset
077 C056 7E                     MOV   A,M
078 C057 23                     INX   H
079 C058 66                     MOV   H,M
080 C059 6F                     MOV   L,A       Get addr routine in HL
081 C05A D1                     POP   D
082 C05B F1                     POP   PSW
083 C05C E3                     XTHL            New addr on stack
084 C05D C9                     RET             Continu with new address
085                     *
086                     ********************
087                     * ARGUMENT ERROR *
088                     ********************
089                     *
090                     * Jump to (00D0/1)+2 = Address 'number out of range'
091                     * routine (C7F4).
092                     *
093                     * Entry/exit: See FPEOV.
094                     *
095 C05E E5             FPEAE   PUSH  H
096 C05F 210200                 LXI   H,:0002   Init. offset
097 C062 C34FC0                 JMP   :C04F     Calc. new addr, go to it
098                     *
099                     ********************
100                     * UNDERFLOW ERROR *
101                     ********************
102                     *
103                     * Jump to (00D0/1)+4 = Return (C7F6).
104                     * Underflow gives 0 as result of operation.
105                     *
106                     * Entry/exit: See FPEOV.
107                     *
108 C065 E5             FPEUN   PUSH  H
109 C066 215EC4                 LXI   H,:C45E   Addr. FPT(0)
110 C069 C30CD2                 JMP   :D20C     Copy '0' into MACC
111                     *
112                     ************************
113                     * DIVIDE BY ZERO ERROR *
114                     ************************
115                     *
116                     * Jump to (00D0/1)+6 = Address 'divide by zero'
117                     * routine (C7F8).
118                     *
119                     * Entry/exit: See FPEOV.
120                     *
121 C06C E5             FPEDO   PUSH  H
122 C06D 210600                 LXI   H,:0006   Init. offset
123 C070 C34FC0                 JMP   :C04F     Calc. new addr, go to it
124                     *
125                     *
```

```
126                     ****************************
127                     * GET CHARACTER FROM LINE *
128                     ****************************
129                     *
130                     * Entry: None.
131                     * Exit : All registers preserved.
132                     *        Address to continue on stack.
133                     *
134 C073 E5             LC22    PUSH  H
135 C074 2AD200                 LHLD  :00D2     Get addr 'Get char' routine
136 C077 E3                     XTHL            on stack; restore HL
137 C078 C9                     RET             Goto (00D0/1)+2
138                     *
139                     ****************************
140                     * FLOATING POINT COMPARE *
141                     ****************************
142                     *
143                     * Compares normalised FPT numbers in MACC
144                     * and in M.
145                     *
146                     * Exit: ABCDEHL preserved.
147                     *       Flags: CY=1,S=0,Z=1: both nrs. 0
148                     *              CY=0,S=0,Z=1: both nrs. identical
149                     *              CY=0,S=0,Z=0: MACC > M
150                     *              CY=0,S=1,Z=0: MACC < M
151                     *
152 C079 C5             FCOMP   PUSH  B
153 C07A F5                     PUSH  PSW
154 C07B D5                     PUSH  D
155 C07C E5                     PUSH  H
156 C07D E7                     RST   4         Copy MACC to reg A,B,C,D
157 C07E 15                     DATA  :15
158 C07F 5F                     MOV   E,A       Exp.byte in E
159 C080 AE                     XRA   M         XOR both exp.bytes
160 C081 FAB7C0                 JM    :C0B7     Jump if different signs
161
162                     * If equal signs:
163
164 C084 C3E8D1                 JMP   :D1E8     Goto D1E8, return to C087
165                     *
166 C087 17             LC23    RAL
167 C088 C2A3C0                 JNZ   :C0A3
168 C08B 7B             LC24    MOV   A,E
169 C08C 96                     SUB   M         Comp. exp. bytes
170 C08D C2A2C0                 JNZ   :C0A2     Jump if not equal
171 C090 23                     INX   H
172 C091 78                     MOV   A,B
173 C092 96                     SUB   M         Comp. 1st bytes mantissa's
174 C093 C2A2C0                 JNZ   :C0A2     Jump if not equal
175 C096 23                     INX   H
176 C097 79                     MOV   A,C
177 C098 96                     SUB   M         Comp. 2nd bytes mantissa's
178 C099 C2A2C0                 JNZ   :C0A2     Jump if not equal
179 C09C 23                     INX   H
180 C09D 7A                     MOV   A,D
181 C09E 96                     SUB   M         Comp. 3rd bytes mantissa's
182 C09F CAA6C0         LC25    JZ    :C0A6     Jump if not equal
183 C0A2 1F             LC26    RAR             ) Set flags for output
184 C0A3 AB             LC27    XRA   E         )
185 C0A4 F601           LC28    ORI   :01       Clear CY-flag
186 C0A6 E1             LC29    POP   H
187 C0A7 D1                     POP   D
```

```
                    '       POP   B
C0A8 C1             '       MOV   A,B       Restore A
C0A9 78                     POP   B
C0AA C1                     RET
C0AB C9             *
                    ********************
                    * INTEGER COMPARE *
                    ********************
                    *
                    * Compares INT numbers in MACC and M.
                    * REMARK: Routine is incorrect when both
                    *         numbers are negative ! Then result
                    *         is if MACC > M due to LC26/LC27.
                    *
                    * Exit:  ABCDEHL preserved. CY=0
                    *        Flags: S=0, Z=1: Both numbers equal
                    *               S=0, Z=0: MACC > M
                    *               S=1, Z=0: MACC < M
                    *
C0AC C5             ICOMP   PUSH  B
C0AD F5                     PUSH  PSW
C0AE D5                     PUSH  D
C0AF E5                     PUSH  H
C0B0 E7                     RST   4         Copy MACC to reg. A,B,C,D
C0B1 15                     DATA  :15
C0B2 5F                     MOV   E,A       Sign byte in E
C0B3 AE                     XRA   M         XOR both sign bytes
C0B4 F28BC0                 JP    :C08B     If both nrs have same sign:
                                              compare

                    * If different signs:

C0B7 AE             LC241   XRA   M         Find out which one is neg:
                                              S=1: MEM pos; MACC neg
                                              S=0: MEM neg; MACC pos
C0B8 C3A4C0                 JMP   :C0A4     Abort
                    *
                    ****************************************
                    * INCREMENT INTEGER NUMBER IN MEMORY *
                    ****************************************
                    *
                    * Entry: HL points to 1st byte of INT number.
                    * Exit:  All registers preserved.
                    *
C0BB F5             IINM    PUSH  PSW
C0BC E5                     PUSH  H
C0BD 23                     INX   H
C0BE 23                     INX   H
C0BF 23                     INX   H         HL pnts to last byte
C0C0 3E03                   MVI   A,:03     Nr of bytes for INT nr
C0C2 34             LC30    INR   M         Incr. INT nr.
C0C3 C2D2C0                 JNZ   :C0D2     Ready if no overflow
C0C6 2B                     DCX   H         Goto next byte
C0C7 3D                     DCR   A         1st byte reached?
C0C8 C2C2C0                 JNZ   :C0C2     Incr. next byte
C0CB 34                     INR   M         Incr. 1st byte
C0CC 7E                     MOV   A,M       Get it
C0CD FE80                   CPI   :80       msb=1?
C0CF CC4BC0                 CZ    :C04B     Then overflow error
C0D2 E1             LC31    POP   H         Normal return
C0D3 F1                     POP   PSW
C0D4 C9                     RET
```

```
                        *
                        ********************************************************
                        * DECREMENT INTEGER NUMBER IN MEMORY - (not used) *
                        ********************************************************
                        *
                        * Entry: HL points to 1st byte of INT number.
                        * Exit:  All registers preserved.
                        *
C0D5 F5         IDCM    PUSH  PSW
C0D6 C5                 PUSH  B
C0D7 E5                 PUSH  H
C0D8 23                 INX   H
C0D9 23                 INX   H
C0DA 23                 INX   H             HL pnts to last byte
C0DB 0603               MVI   B,:03         Nr. of bytes of INT nr.
C0DD 35         LC32    DCR   M             Decr. INT nr
C0DE 7E                 MOV   A,M
C0DF 3C                 INR   A             Check for overflow
C0E0 C2EFC0             JNZ   :C0EF         Ready if no overflow
C0E3 2B                 DCX   H             Goto next byte
C0E4 05                 DCR   B             Decr. byte count
C0E5 C2DDC0             JNZ   :C0DD         Next byte if not ready
C0E8 35                 DCR   M             Decr. hibyte
C0E9 7E                 MOV   A,M
C0EA FE7F               CPI   :7F           Check for overflow
C0EC CC4BC0             CZ    :C04B         Then run overflow error
C0EF E1         LC33    POP   H             Normal return
C0F0 C1                 POP   B
C0F1 F1                 POP   PSW
C0F2 C9                 RET
                        *
                        ************************************************
                        * INCREMENT FLOATING POINT NUMBER IN MEMORY *
                        ************************************************
                        *
                        * If number = 0, or exponent < 0, a '1' is added
                        * to the lsb of the mantissa. Else, the position
                        * of the least significant '1' is looked up, and
                        * a '1' is added to this position.
                        * If the lsb of the mantissa is already a rounded
                        * value, no increment occurs.
                        *
                        * Entry: HL points to 1st byte of FPT number.
                        * Exit:  All registers preserved.
                        *
C0F3 F5         FINM    PUSH  PSW
C0F4 C5                 PUSH  B
C0F5 D5                 PUSH  D
C0F6 E5                 PUSH  H
C0F7 1162C4             LXI   D,:C462       Addr FPT(1)
C0FA 7E                 MOV   A,M           Get exp.byte
C0FB E67F               ANI   :7F           Mask sign bit
C0FD CAACC1             JZ    :C1AC         If nr=0: add 1, abort
C100 FE40               CPI   :40           Is exponent negative?
C102 D2ACC1             JNC   :C1AC         Then add 1, abort
C105 FE19               CPI   :19           Lsb of mantissa is not lsb
                                            of number ?
C107 D24DC1             JNC   :C14D         Then Popall, ret
C10A BE                 CMP   M             Check if nr. is negative
C10B 23                 INX   H
C10C C252C1             JNZ   :C152         Then jump
```

```
                        * From LC34 also used by XFDCM.
                        * Find lsb of mantissa if nr is positive:
                        
C10F D609       LC34    SUI   :09       In 1st byte ?
C111 DCEEC1             CC    :C1EE     Then SHL bit into A (A) time
C114 DA36C1             JC    :C136     and jump
C117 23                 INX   H
C118 D608               SUI   :08       In 2nd byte ?
C11A DCEEC1             CC    :C1EE     Then SHL bit into A (A) time
C11D DA2EC1             JC    :C12E     and jump
C120 23                 INX   H
C121 D608               SUI   :08       In 3rd byte ?
C123 CDEEC1             CALL  :C1EE     Then SHL bit into A (A) time
C126 86                 ADD   M
C127 77                 MOV   M,A       Add 1 to 3rd byte mantissa
C128 D24DC1             JNC   :C14D     Ready if no overflow
C12B 2B                 DCX   H
C12C 3E01               MVI   A,:01     Overflow: add 1 to 2nd byte
C12E 86         LC35    ADD   M
C12F 77                 MOV   M,A       Add 1 to 2nd byte mantissa
C130 D24DC1             JNC   :C14D     Ready if no overflow
C133 2B                 DCX   H
C134 3E01               MVI   A,:01     Overflow: add 1 to 1st byte
C136 86         LC36    ADD   M
C137 77                 MOV   M,A       Add 1 to 1st byte mantissa
C138 D24DC1             JNC   :C14D     Ready if no overflow

                        * If overflow into exponent byte:

C13B 1F                 RAR             )
C13C 77                 MOV   M,A       )
C13D 23                 INX   H         )
C13E 7E                 MOV   A,M       ) Shift all bits in
C13F 1F                 RAR             ) mantissa right
C140 77                 MOV   M,A       ) one position
C141 23                 INX   H         )
C142 7E                 MOV   A,M       )
C143 1F                 RAR             )
C144 77                 MOV   M,A       )
C145 2B                 DCX   H
C146 2B                 DCX   H
C147 2B                 DCX   H         HL pnts to exp.byte
C148 3E01               MVI   A,:01
C14A CDBAC1     LC37    CALL  :C1BA     Add 1 to exponent
                        *
C14D E1         EXIT    POP   H
C14E D1                 POP   D
C14F C1                 POP   B
C150 F1                 POP   PSW
C151 C9                 RET

                        * Find lsb of mantissa if nr is negative:

C152 D609       LC39    SUI   :09       In 1st byte ?
C154 DCEEC1             CC    :C1EE     Then SHL bit into A (A) time
C157 DA7DC1             JC    :C17D     and jump
C15A 23                 INX   H
C15B D608               SUI   :08       In 2nd byte ?
C15D DCEEC1             CC    :C1EE     Then SHL bit into A (A) time
C160 DA73C1             JC    :C173     and jump
C163 23                 INX   H
C164 D608               SUI   :08       In 3rd byte ?
```

```
374 C166 DCEEC1              CC    :C1EE     Then SHL bit into A (A) time
375 C169 47                  MOV   B,A
376 C16A 7E                  MOV   A,M
377 C16B 90                  SUB   B         Subtract 1 from 3rd byte
378 C16C 77                  MOV   M,A
379 C16D D24DC1              JNC   :C14D     Ready if no borrow
380 C170 2B                  DCX   H
381 C171 3E01                MVI   A,:01     Subtract 1 from 2nd byte if
382                                          borrow
383 C173 47          LC40    MOV   B,A
384 C174 7E                  MOV   A,M
385 C175 90                  SUB   B         Subtract 1 from 2nd byte
386 C176 77                  MOV   M,A
387 C177 D24DC1              JNC   :C14D     Ready if no borrow
388 C17A 2B                  DCX   H
389 C17B 3E01                MVI   A,:01     Subtract 1 from 1st byte if
390                                          borrow
391 C17D 47          LC41    MOV   B,A
392 C17E 7E                  MOV   A,M
393 C17F 90                  SUB   B         Subtract 1 from 1st byte
394 C180 77                  MOV   M,A
395 C181 FA4DC1              JM    :C14D     Ready if normalised
396
397                  * If not normalised:
398
399 C184 061B                MVI   B,:18     Nr of mantissa bits
400 C186 23          LC42    INX   H         )
401 C187 23                  INX   H         )
402 C188 B7                  ORA   A         )
403 C189 7E                  MOV   A,M       )
404 C18A 17                  RAL             )
405 C18B 77                  MOV   M,A       ) Shift all bits
406 C18C 2B                  DCX   H         ) of mantissa
407 C18D 7E                  MOV   A,M       ) left one position
408 C18E 17                  RAL             )
409 C18F 77                  MOV   M,A       )
410 C190 2B                  DCX   H         )
411 C191 7E                  MOV   A,M       )
412 C192 17                  RAL             )
413 C193 77                  MOV   M,A       )
414 C194 B7                  ORA   A
415 C195 FA9FC1              JM    :C19F     If normalized
416 C198 05                  DCR   B         Update exp. count
417 C199 CAA6C1              JZ    :C1A6     If exp. now zero
418 C19C C386C1              JMP   :C186     Cont. normalisation
419
420                  * Normalisation done:
421
422 C19F 2B          LC43    DCX   H         Pnts to exp.byte
423 C1A0 78                  MOV   A,B       Get exp. count
424 C1A1 D619                SUI   :19       Minus nr of bytes in
425                                          mantissa
426 C1A3 C34AC1              JMP   :C14A     Update exponent, quit
427
428                  * If exponent is zero:
429
430 C1A6 2B          LC44    DCX   H
431 C1A7 3600                MVI   M,:00     Exp.byte is 0
432 C1A9 C34DC1              JMP   :C14D     Popall, ret
433
434                  * Simply add 1 (FINM) or add -1 (FDCM):
435
```

```
C1AC E7          LC45    RST   4          Copy number into MACC
C1AD 0C                  DATA  :0C
C1AE EB                  XCHG
C1AF E7                  RST   4          Add 1 or -1 (FPT)
C1B0 00                  DATA  :00
C1B1 EB                  XCHG
C1B2 E7                  RST   4          Copy MACC into memory
C1B3 0F                  DATA  :0F
C1B4 C34DC1              JMP   :C14D      Popall, ret
                 *
                 ******************
                 * ADD EXPONENTS *
                 ******************
                 *
                 * LC225 for operand in MACC.
                 * LC46 for operand in M.
                 *
                 * Entry: Byte to be added to exponent in A.
                 * Exit:  BCDEHL preserved.
                 *        CY=0: O.K.
                 *        CY=1: Overflow.
                 *
C1B7 21D500      LC225   LXI   H,:00D5    Addr. MACC
                 *
C1BA E5          LC46    PUSH  H
C1BB D5                  PUSH  D
C1BC C5                  PUSH  B
C1BD 4F                  MOV   C,A        Byte to be added in C
C1BE 7E                  MOV   A,M        Get exp.byte operand
C1BF E680                ANI   :80        Sign bit mantissa only
C1C1 47                  MOV   B,A        in B
C1C2 7E                  MOV   A,M        Get exp.byte
C1C3 CDE9C1              CALL  :C1E9      Sign extend
C1C6 F5                  PUSH  PSW        Save sign extended exp.byte
C1C7 A9                  XRA   C          XOR with byte to be added
C1C8 2F                  CMA
C1C9 57                  MOV   D,A
C1CA F1                  POP   PSW        Get sign extended exp.byte
C1CB 81                  ADD   C          Add byte to exponent
C1CC 4F                  MOV   C,A        Store result
C1CD 1F                  RAR
C1CE A9                  XRA   C
C1CF A2                  ANA   D
C1D0 FAE2C1              JM    :C1E2      If overflow into sign bit
C1D3 79                  MOV   A,C        Get new exp.byte
C1D4 17                  RAL
C1D5 A9                  XRA   C
C1D6 FAE2C1              JM    :C1E2      If overflow into sign bit
C1D9 79                  MOV   A,C        Get new exp.byte
C1DA E67F                ANI   :7F        Exponent only
C1DC B0                  ORA   B          Add sign bit mantissa
C1DD 77                  MOV   M,A        Store it
C1DE C1          LC47    POP   B
C1DF D1                  POP   D
C1E0 E1                  POP   H
C1E1 C9                  RET              CY=0

                 * If overflow into sign bit:

C1E2 79          LC48    MOV   A,C        Get new exp.byte
C1E3 17                  RAL
C1E4 B7                  ORA   A
```

```
498 C1E5 37                 STC
499 C1E6 C3DEC1             JMP   :C1DE     Abort with CY=1
500                 *
501                 ***************
502                 * SIGN EXTEND *
503                 ***************
504                 *
505                 * Exponent byte is normalized.
506                 *
507                 * Entry: Exp.byte in A.
508                 * Exit:  BCDEHL preserved.
509                 *        Normalized exp.byte in A: Exp.value in
510                 *        bits 7-2, sign mantissa in bit 1, sign
511                 *        exponent in bit 0.
512                 *
513 C1E9 07        SEXT    RLC
514 C1EA 07                RLC
515 C1EB 07                RLC
516 C1EC 1F                RAR
517 C1ED C9                RET
518                 *
519                 ************************************
520                 * MOVE A BIT INTO A LEFT (A) TIMES *
521                 ************************************
522                 *
523                 * Used to place a '1' in the correct position in
524                 * a byte for adding/subtracting '1' to/from the
525                 * least significant '1' of a FPT mantissa.
526                 *
527                 * Entry: A contains a neg. number indicating
528                 *        how often RAL has to be performed.
529                 * Exit:  Result in A,B.
530                 *        FCDEHL preserved.
531                 *
532 C1EE F5        LC50    PUSH  PSW
533 C1EF 47                MOV   B,A       Save nr of shifts
534 C1F0 AF                XRA   A         Clear A
535 C1F1 37                STC             Set CY
536 C1F2 17        LC51    RAL             SHL
537 C1F3 04                INR   B         Update count
538 C1F4 C2F2C1            JNZ   :C1F2     Continu if not ready
539 C1F7 47                MOV   B,A       Save result
540 C1F8 F1                POP   PSW
541 C1F9 78                MOV   A,B       Result in A
542 C1FA C9                RET
543                 *
544                 *
545                 *
546 C1FB                   END
```

```
****************************
* S Y M B O L   T A B L E *
****************************

BASE   C000   DECBUF C033   EXIT   C14D   FCOMP  C079
FINM   C0F3   FPEAE  C05E   FPEDO  C06C   FPEOV  C04B
FPEUN  C065   ICOMP  C0AC   IDCM   C0D5   IINM   C0BB
LC18   C047   LC20   C04F   LC22   C073   LC225  C1B7
LC23   C087   LC24   C08B   LC241  C0B7   LC25   C09F
LC26   C0A2   LC27   C0A3   LC28   C0A4   LC29   C0A6
LC30   C0C2   LC31   C0D2   LC32   C0DD   LC33   C0EF
LC34   C10F   LC35   C12E   LC36   C136   LC37   C14A
```

LC39	C152	LC40	C173	LC41	C17D	LC42	C186
LC43	C19F	LC44	C1A6	LC45	C1AC	LC46	C1BA
LC47	C1DE	LC48	C1E2	LC50	C1EE	LC51	C1F2
MINIT	C035	SEXT	C1E9	XFBC	C021	XFCB	C01E
XFCOMP	C00C	XFDCM	C009	XFINM	C006	XHBC	C02D
XHCB	C02A	XIBC	C027	XICB	C024	XICOMP	C015
XIDCM	C012	XIINM	C00F	XINIT	C003	XPOP	C01B
XPRTY	C030	XPUSH	C018				

```
002                             ORG   :C1FB
003                     *
004                     *
005                     *
006                     ***********************************************
007                     * DECREMENT FLOATING POINT NUMBER IN MEMORY *
008                     ***********************************************
009                     *
010                     * Routine is not used.
011                     *
012                     * If the number is 0, or the exponent < 0, -1 is
013                     * added to the mantissa. Else, a -1 is added/
014                     * subtracted to/from the least significant '1' of
015                     * the mantissa.
016                     * If the lsb of the mantissa is already a rounded
017                     * value, no decrement occurs.
018                     *
019                     * Entry: HL points to FPT number in M.
020                     * Exit:  All registers preserved.
021                     *
022 C1FB F5             FDCM    PUSH  PSW
023 C1FC C5                     PUSH  B
024 C1FD D5                     PUSH  D
025 C1FE E5                     PUSH  H
026 C1FF 111AC2                 LXI   D,:C21A     Addr. FPT(-1)
027 C202 7E                     MOV   A,M         Get exp.byte
028 C203 E67F                   ANI   :7F         Mask sign bit
029 C205 CAACC1                 JZ    :C1AC       If nr=0: add -1, abort
030 C208 FE40                   CPI   :40         Is exp. negative ?
031 C20A D2ACC1                 JNC   :C1AC       Then add -1, abort
032 C20D FE18                   CPI   :18         Max. nr of mantissa bits
033 C20F D24DC1                 JNC   :C14D       Abort if lsb mantissa is
034                                               not lsb of number
035 C212 BE                     CMP   M           Check if nr. is negative
036 C213 23                     INX   H
037 C214 CA52C1                 JZ    :C152       Into FINM for neg. nr
038 C217 C30FC1                 JMP   :C10F       Idem for pos. nr.
039                     *
040                     * DATA - (not used):
041                     *
042 C21A 81             FPM1    DATA  :81         FPT (-1)
043 C21B 80                     DATA  :80
044 C21C 00                     DATA  :00
045 C21D 00                     DATA  :00
046                     *
047                     ************************
048                     * SAVE MACC ON STACK *
049                     ************************
050                     *
051                     * Contents MACC is placed on TOS. Returnaddress
052                     * is saved.
053                     *
054                     * Entry: None.
055                     * Exit:  All registers preserved.
056                     *        On stack: HL; returnaddress; MACC.
057                     *
058 C21E 22E100         PUSH    SHLD  :00E1       Save HL
059 C221 E3                     XTHL              Get returnaddress
060 C222 22DF00                 SHLD  :00DF       Save it
061 C225 E5                     PUSH  H           and put it on stack again
062 C226 210000                 LXI   H,:0000
063 C229 39                     DAD   SP          SP in HL
```

```
064 C22A E7              RST   4          Copy MACC to TOS
065 C22B 0F              DATA  :0F
066 C22C 2ADF00          LHLD  :00DF      Get returnaddr.
067 C22F E5              PUSH  H          on stack
068 C230 2AE100    LC52  LHLD  :00E1      Get original HL
069 C233 C9              RET
070                *
071                ******************************
072                * RETRIEVE MACC FROM STACK *
073                ******************************
074                *
075                * Gets data from TOS and place it in MACC.
076                *
077                * Entry: None.
078                * Exit:  All registers preserved.
079                *
080 C234 22E100    POP   SHLD  :00E1      Save HL
081 C237 E1              POP   H          Get returnaddress
082 C238 22DF00          SHLD  :00DF      Save it
083 C23B 210000          LXI   H,:0000
084 C23E 39              DAD   SP         Get SP in HL
085 C23F E7              RST   4          Copy TOS to MACC
086 C240 0C              DATA  :0C
087 C241 E1              POP   H
088 C242 2ADF00          LHLD  :00DF      Get returnaddress
089 C245 E3              XTHL             on stack
090 C246 C330C2          JMP   :C230      Restore HL, ret
091                *
092                ******************************************
093                * INPUT A FLOATING POINT NUMBER TO MACC *
094                ******************************************
095                *
096                * Converts a FPT number to binary into MACC.
097                * The input string is converted as a integer FPT
098                * number, then multiplied/divided by a power of
099                * 10, corresponding to the explicit exponent and
100                * placement of the decimal point.
101                *
102                * Entry: C points to 1st digit of FPT nr in input.
103                * Exit:  CY=1: No error.
104                *        CY=0: Over/underflow error.
105                *        C points past FPT string in input.
106                *        ABDEHL, rest of F preserved.
107                *
108 C249 37        FCB   STC              CY=1
109 C24A F5              PUSH  PSW
110 C24B D5              PUSH  D
111 C24C E5              PUSH  H
112 C24D CDAEC2          CALL  :C2AE      Clear MACC+DEH; L=2B
113 C250 CD2FC3    LC53  CALL  :C32F      Get bin.value of input char
114 C253 DCBAC2          CC    :C2BA      Value found: Move digit into
115                                       MACC
116 C256 DA50C2          JC    :C250      and get next digit
117 C259 FE2E            CPI   :2E        '.' ?
118 C25B CA6BC2          JZ    :C26B      Then jump
119 C25E 1D              DCR   E
120 C25F 1C              INR   E
121 C260 CAA6C2          JZ    :C2A6      If error
122 C263 FE45            CPI   :45        'E' ?
123 C265 CA84C2          JZ    :C284      Then jump
124 C268 C39FC2          JMP   :C29F      Convert FPT exp; quit
125
```

```
126                     * If digit is '.':
127
128 C26B CDD5C2         LC54    CALL    :C2D5       E=0, H=H+1
129 C26E CD2FC3         LC55    CALL    :C32F       Get bin.value of input char
130 C271 DCBAC2                 CC      :C2BA       If found: Move digit into
131                                                 MACC
132 C274 DA6EC2                 JC      :C26E       and get next digit
133 C277 1D                     DCR     E
134 C278 1C                     INR     E
135 C279 CAA6C2                 JZ      :C2A6       If error
136 C27C FE45                   CPI     :45         'E' ?
137 C27E C4D9C2                 CNZ     :C2D9       H=0 if not
138 C281 C29FC2                 JNZ     :C29F       If not: convert exp, quit
139
140                     * If digit is 'E':
141
142 C284 CDD9C2         LC56    CALL    :C2D9       H=0
143 C287 CD2FC3                 CALL    :C32F       Get bin.value of input char
144 C28A CCDCC2                 CZ      :C2DC       If '+' or '-': char in L
145 C28D CC2FC3                 CZ      :C32F       Then get bin.value of next
146                                                 char
147 C290 D2A6C2                 JNC     :C2A6       Error if no char found
148 C293 CDDEC2                 CALL    :C2DE       H = 10 * H + A
149 C296 CD2FC3                 CALL    :C32F       Get bin.value of input char
150 C299 DCDEC2                 CC      :C2DE       If found: H = 10 * H + A
151 C29C DC2FC3                 CC      :C32F       Get bin.value of input char
152
153                     * If digit is number:
154
155 C29F CDEBC2         LC57    CALL    :C2EB       Convert FPT exponent
156 C2A2 E1             LC58    POP     H
157 C2A3 D1                     POP     D
158 C2A4 F1                     POP     PSW         CY=1
159 C2A5 C9                     RET
160
161                     * If error:
162
163 C2A6 CD2DC3         LC59    CALL    :C32D       DCR C
164 C2A9 E1             LC60    POP     H
165 C2AA D1                     POP     D
166 C2AB F1                     POP     PSW
167 C2AC 3F                     CMC                 CY=0
168 C2AD C9                     RET
169                     *
170                     * CLEAR MACC AND REGISTERS D, E AND H:
171                     *
172                     * Exit:  ABC preserved.
173                     *        L = 2B ('+')
174                     *
175 C2AE 215EC4         LC61    LXI     H,:C45E     Addr. FPT(0)
176 C2B1 E7                     RST     4           Copy FPT(0) to MACC
177 C2B2 0C                     DATA    :0C
178 C2B3 110000                 LXI     D,:0000     Clear DE
179 C2B6 212B00                 LXI     H,:002B     Clear H, L='+'
180 C2B9 C9                     RET
181                     *
182                     * MOVE A DIGIT INTO THE MACC:
183                     *
184                     * MACC = MACC * 10 + A.
185                     *
186                     * Entry: A: Digit 1 - 9.
187                     * Exit:  AFBCHL preserved.
```

```
188                     *       D=D-H; E=E-1.
189                     *
190 C2BA F5             LC62    PUSH  PSW
191 C2BB E5                     PUSH  H
192 C2BC 214DC3                 LXI   H,:C34D   Addr FPT (10)
193 C2BF E7                     RST   4         MACC = MACC * 10 (FPT)
194 C2C0 06                     DATA  :06
195 C2C1 D5                     PUSH  D
196 C2C2 87                     ADD   A         )
197 C2C3 87                     ADD   A         ) DE = 4 * A
198 C2C4 5F                     MOV   E,A       ) (calc offset to startaddr)
199 C2C5 1600                   MVI   D,:00     )
200 C2C7 215EC4                 LXI   H,:C45E   Addr table FPT(1-9)
201 C2CA 19                     DAD   D         Calc. addr nr to be added
202 C2CB D1                     POP   D
203 C2CC E7                     RST   4         MACC = MACC + (1-9) (FPT)
204 C2CD 00                     DATA  :00
205 C2CE E1                     POP   H
206 C2CF 7A                     MOV   A,D
207 C2D0 94                     SUB   H
208 C2D1 57                     MOV   D,A       D=D-H
209 C2D2 1D                     DCR   E         E=E-1
210 C2D3 F1                     POP   PSW
211 C2D4 C9                     RET
212                     *
213 C2D5 1E00           LC63    MVI   E,:00
214 C2D7 24                     INR   H
215 C2D8 C9                     RET
216                     *
217 C2D9 2600           LC64    MVI   H,:00
218 C2DB C9                     RET
219                     *
220 C2DC 6F             LC65    MOV   L,A
221 C2DD C9                     RET
222                     *
223                     * H = 10 * H + A:
224                     *
225                     * Exit: AFBCDEL preserved.
226                     *
227 C2DE F5             LC66    PUSH  PSW
228 C2DF 7C                     MOV   A,H       A=H
229 C2E0 87                     ADD   A         A=2*H
230 C2E1 87                     ADD   A         A=4*H
231 C2E2 84                     ADD   H         A=5*H
232 C2E3 87                     ADD   A         A=10*H
233 C2E4 67                     MOV   H,A
234 C2E5 F1                     POP   PSW
235 C2E6 F5                     PUSH  PSW
236 C2E7 84                     ADD   H         A=10*H+A
237 C2E8 67                     MOV   H,A
238 C2E9 F1                     POP   PSW
239 C2EA C9                     RET
240                     *
241                     * CONVERT A FPT EXPONENT:
242                     *
243                     * The MACC is multiplied/divided by a power of 10
244                     * corresponding to the 'E'-exponent minus the number
245                     * of digits after the decimal point.
246                     *
247                     * Entry: C:    Points beyond 1st nonuseable char of
248                     *              FPT number in input.
249                     *       L:     Contains sign of exponent.
```

```
250                     *         H:    Contains 'E....' exponent (10).
251                     *         MACC: Contains FPT conversion of string of
252                     *               digits.
253                     *         D:    Contains nr of digits after '.'.
254                     * Exit:   BE preserved, AHL corrupted.
255                     *         C:    Decremented to after FPT nr in input.
256                     *         D:    Contains effective exponent.
257                     *
258 C2EB CD2DC3      LC67    CALL  :C32D       Decr C
259 C2EE 7D                  MOV   A,L         Get exp. sign
260 C2EF FE2D                CPI   :2D         '-' ?
261 C2F1 7C                  MOV   A,H         Get exponent
262 C2F2 C2F7C2              JNZ   :C2F7       If exp. positive
263 C2F5 2F                  CMA               ) Else: make exponent
264 C2F6 3C                  INR   A           ) positive
265 C2F7 82          LC68    ADD   D           Add nr of digits after '.'
266 C2F8 57                  MOV   D,A         Save result
267 C2F9 F2FEC2              JP    :C2FE       If result positive
268 C2FC 2F                  CMA               ) Else: make it
269 C2FD 3C                  INR   A           ) positive
270 C2FE C5          LC69    PUSH  B
271 C2FF 0605                MVI   B,:05       Nr of times of multipl.
272 C301 214DC3              LXI   H,:C34D     Addr table powers of 10
273 C304 B7          LC70    ORA   A           Flags on result LC68
274 C305 1F                  RAR               lsb in carry
275 C306 D217C3              JNC   :C317       If bit=0
276 C309 F5                  PUSH  PSW
277 C30A 7A                  MOV   A,D         Check if multipl/div
278 C30B B7                  ORA   A
279 C30C FA11C3              JM    :C311       If division
280 C30F E7                  RST   4           MACC = MACC * power of 10
281 C310 06                  DATA  :06
282 C311 CA16C3      LC71    JZ    :C316       If multiplication
283 C314 E7                  RST   4           MACC = MACC / power of 10
284 C315 09                  DATA  :09
285 C316 F1          LC72    POP   PSW         Restore A
286 C317 23          LC73    INX   H
287 C318 23                  INX   H
288 C319 23                  INX   H
289 C31A 23                  INX   H           HL pnts to next ^10
290 C31B 05                  DCR   B
291 C31C C204C3              JNZ   :C304       Again if not ready
292 C31F B7                  ORA   A
293 C320 CA2BC3              JZ    :C32B       If result OK
294
295                     * If error:
296
297 C323 7A                  MOV   A,D
298 C324 B7                  ORA   A           Set flags for error type
299 C325 F44BC0              CP    :C04B       If overflow error
300 C328 FC65C0              CM    :C065       If underflow error
301 C32B C1          LC74    POP   B           Normal return
302 C32C C9                  RET
303                     *
304 C32D 0D          LC75    DCR   C
305 C32E C9                  RET
306                     *
307                     * GET BINARY VALUE OF INPUT CHARACTER IN A:
308                     *
309                     * Entry: C points to character in input.
310                     * Exit:  C points to next character.
311                     *        BDEHL preserved.
```

```
                        *       CY=1, Z=0: Value in A.
                        *       CY=0, Z=1: Char is +/-.
                        *       CY=0, Z=0: otherwise.
                        *
C32F CD73C0             LC76    CALL  :C073     Get char from line
C332 0C                         INR   C         Update pointer
C333 FE2B                       CPI   :2B
C335 C8                         RZ              Abort if '+'
C336 FE2D                       CPI   :2D
C338 C8                         RZ              Abort if '-'
C339 FE30                       CPI   :30
C33B 3F                         CMC
C33C D0                         RNC             Abort if < #30
C33D FE3A                       CPI   :3A
C33F D24BC3                     JNC   :C34B     Abort if > #3A
C342 D630                       SUI   :30       Convert ASCII to binary
C344 D5                         PUSH  D
C345 57                         MOV   D,A
C346 3C                         INR   A         Set Z-flag correctly for
                                                reqd output
C347 7A                         MOV   A,D
C348 D1                         POP   D
C349 37                         STC             CY=1: value in A
C34A C9                         RET
C34B B7                 LC77    ORA   A         Set flags correctly
C34C C9                         RET
                        *
                        ****************************
                        * TABLE FPT POWERS OF 10 *
                        ****************************
                        *
C34D 04                 LC233   DATA  :04       FPT 10^1
C34E A0                         DATA  :A0
C34F 00                         DATA  :00
C350 00                         DATA  :00
                        *
C351 07                         DATA  :07       FPT 10^2
C352 C8                         DATA  :C8
C353 00                         DATA  :00
C354 00                         DATA  :00
                        *
C355 0E                         DATA  :0E       FPT 10^4
C356 9C                         DATA  :9C
C357 40                         DATA  :40
C358 00                         DATA  :00
                        *
C359 1B                         DATA  :1B       FPT 10^8
C35A BE                         DATA  :BE
C35B BC                         DATA  :BC
C35C 20                         DATA  :20
                        *
C35D 36                         DATA  :36       FPT 10^16
C35E 8E                         DATA  :8E
C35F 1B                         DATA  :1B
C360 CA                         DATA  :CA
                        *
                        ************************************************
                        * CONVERT A FLOATING POINT NUMBER FOR OUTPUT *
                        ************************************************
                        *
                        * A FPT number in MACC is converted to ASCII in
                        * outputbuffer 00E4-F1. The sign is in 00E4, the
```

```
374                     * decimal point in 00E5. The normalized value of
375                     * the mantissa is in 00E6-EC (7 digits). In 00F1
376                     * is the 10's exponent in 2-complement binairy
377                     * signed format.
378                     *
379                     * Exit: A=6 (number of significant digits).
380                     *       BCDEHL, MACC preserved.
381                     *
382 C361 C5            FBC     PUSH  B
383 C362 D5                    PUSH  D
384 C363 E5                    PUSH  H
385 C364 CD1EC2                CALL  :C21E     Save MACC on stack
386 C367 E7                    RST   4         Copy MACC to reg A,B,C,D
387 C368 15                    DATA  :15
388 C369 F5                    PUSH  PSW       Save exp.byte
389 C36A B0                    ORA   B
390 C36B B1                    ORA   C
391 C36C B2                    ORA   D
392 C36D CAD4C3                JZ    :C3D4     If FPT nr is zero
393 C370 F1                    POP   PSW       Get exp.byte
394 C371 F5                    PUSH  PSW
395 C372 2600                  MVI   H,:00
396 C374 E67F                  ANI   :7F       Mask sign bit mantissa
397 C376 FE40                  CPI   :40
398 C378 DA80C3                JC    :C380     If exp is positive
399 C37B 25                    DCR   H
400 C37C 2F                    CMA             ) Else: convert
401 C37D E67F                  ANI   :7F       ) exponent to
402 C37F 3C                    INR   A         ) positive
403 C380 F5            LC78    PUSH  PSW       Save value exponent
404 C381 AF                    XRA   A
405 C382 E7                    RST   4         Copy mantissa to MACC
406 C383 12                    DATA  :12
407 C384 F1                    POP   PSW       Get exp.value
408 C385 44                    MOV   B,H       B=FF (exp.<0), 00 (exp.>0)
409 C386 2137C4                LXI   H,:C437   Addr table powers FPT(2)
410 C389 0E00                  MVI   C,:00
411 C38B 1607                  MVI   D,:07     7 digits to be examined
412 C38D 0F            LC79    RRC             Shift exp. into carry
413 C38E F5                    PUSH  PSW       Save rest of exp.
414 C38F 7E                    MOV   A,M       Get 10's power byte
415 C390 23                    INX   H         Points to next
416 C391 D2A2C3                JNC   :C3A2     If n-th power of 2=0:
417                                            go to next
418 C394 81                    ADD   C
419 C395 4F                    MOV   C,A       Total 10's power in C
420 C396 05                    DCR   B
421 C397 04                    INR   B
422 C398 C29DC3                JNZ   :C39D     Jump if exp. negative
423 C39B E7                    RST   4         Multipl. mantissa by
424 C39C 06                    DATA  :06       (2^2^n)/10^m
425 C39D CAA2C3        LC80    JZ    :C3A2
426 C3A0 E7                    RST   4         Divide mantissa by
427 C3A1 09                    DATA  :09       (2^2^n)/10^m
428 C3A2 23            LC81    INX   H
429 C3A3 23                    INX   H
430 C3A4 23                    INX   H
431 C3A5 23                    INX   H         Pnts to next in table
432 C3A6 F1                    POP   PSW       Get rest exponent
433 C3A7 15                    DCR   D         Decr digit count
434 C3A8 C28DC3                JNZ   :C38D     Again if not 7 digits done
435 C3AB 05                    DCR   B
```

```
436 C3AC 04                 INR   B
437 C3AD 215AC4             LXI   H,:C45A   Addr FPT (0.1)
438 C3B0 C2C4C3             JNZ   :C3C4     If exp. negative
439
440                 * If exponent positive:
441
442 C3B3 E5        LC82    PUSH  H
443 C3B4 2162C4             LXI   H,:C462   Addr FPT(1)
444 C3B7 CD79C0             CALL  :C079     Compare with 1
445 C3BA E1                 POP   H
446 C3BB FAD4C3             JM    :C3D4     Jump if normalized
447 C3BE E7                 RST   4         MACC = MACC * 0.1 (FPT)
448 C3BF 06                 DATA  :06
449 C3C0 0C                 INR   C         Update 10's power
450 C3C1 C3B3C3             JMP   :C3B3     Cont. normalisation
451
452                 * If exponent negative:
453
454 C3C4 79        LC83    MOV   A,C       )
455 C3C5 2F                 CMA             ) Change 10's power
456 C3C6 3C                 INR   A         ) to neg. value
457 C3C7 4F                 MOV   C,A       )
458 C3C8 CD79C0    LC84    CALL  :C079     Compare with 0,1
459 C3CB F2D4C3             JP    :C3D4     Jump if normalized
460 C3CE E7                 RST   4         MACC = MACC / 0.1 (FPT)
461 C3CF 09                 DATA  :09
462 C3D0 0D                 DCR   C         Update 10's power
463 C3D1 C3C8C3             JMP   :C3C8     Cont. normalisation
464
465                 * Load output buffer:
466
467 C3D4 79        LC85    MOV   A,C       Get 10's power
468 C3D5 32F100              STA   :00F1     In output buffer
469 C3D8 F1                 POP   PSW       Get signbyte mantissa
470 C3D9 B7                 ORA   A         Set flags on it
471 C3DA 21E400             LXI   H,:00E4   Addr output buffer
472 C3DD 362B               MVI   M,:2B     '+' in buffer
473 C3DF F2E4C3             JP    :C3E4     If mantissa is positive
474 C3E2 362D               MVI   M,:2D     Else: '-' in buffer
475 C3E4 23        LC86    INX   H
476 C3E5 362E               MVI   M,:2E     '.' in 00E5
477 C3E7 23                 INX   H
478 C3E8 E5                 PUSH  H
479 C3E9 E7                 RST   4         Copy MACC to reg A,B,C,D
480 C3EA 15                 DATA  :15
481 C3EB F5                 PUSH  PSW       Save exp.byte
482 C3EC AF                 XRA   A
483 C3ED E7                 RST   4         Copy mantissa to MACC
484 C3EE 12                 DATA  :12
485 C3EF F1                 POP   PSW       Get exp.byte
486 C3F0 2F                 CMA             )
487 C3F1 3C                 INR   A         ) 2-compl.
488 C3F2 E67F               ANI   :7F       Mask sign bit mantissa
489 C3F4 2110C6             LXI   H,:C610   Addr INT(10)
490 C3F7 E7                 RST   4         MACC = MACC * 10 (INT)
491 C3F8 54                 DATA  :54
492 C3F9 2120C4             LXI   H,:C420   Addr. INT(1)
493 C3FC 3D                 DCR   A
494 C3FD FA02C4             JM    :C402     If exp. converted
495 C400 E7                 RST   4         Shift MACC right
496 C401 72                 DATA  :72
497 C402 F2FCC3    LC88     JP    :C3FC     If not ready
```

```
498 C405 E1                     POP   H
499 C406 E7                     RST   4          Copy MACC to reg A,B,C,D
500 C407 15                     DATA  :15
501 C408 C630                   ADI   :30        Exp.byte in ASCII
502 C40A 77                     MOV   M,A        Into outputbuffer
503 C40B 23                     INX   H
504 C40C 0606                   MVI   B,:06      6 sign. digits for mantissa
505 C40E CD24C4       LC89      CALL  :C424      Convert one digit to ASCII
506 C411 77                     MOV   M,A        Into output buffer
507 C412 23                     INX   H
508 C413 05                     DCR   B
509 C414 C20EC4                 JNZ   :C40E      Next digit if not ready
510 C417 CD34C2                 CALL  :C234      Retrieve MACC from TOS
511 C41A E1                     POP   H
512 C41B D1                     POP   D
513 C41C C1                     POP   B
514 C41D 3E06                   MVI   A,:06      6 sign. digits in outputbuf
515 C41F C9                     RET
516                   *
517                   * DATA:
518                   *
519 C420 00           I1        DATA  :00        INT (1)
520 C421 00                     DATA  :00
521 C422 00                     DATA  :00
522 C423 01                     DATA  :01
523                   *
524                   * CONVERT A DIGIT FOR OUTPUT:
525                   *
526                   * Highest byte of MACC *10 is made ASCII.
527                   *
528                   * Exit: A: Converted highest byte MACC.
529                   *       BCHL preserved. DE corrupted.
530                   *
531 C424 C5           LC90      PUSH  B
532 C425 E5                     PUSH  H
533 C426 E7                     RST   4          Copy MACC to reg A,B,C,D
534 C427 15                     DATA  :15
535 C428 AF                     XRA   A          Clear highest byte
536 C429 E7                     RST   4          Copy reg A,B,C,D to MACC
537 C42A 12                     DATA  :12
538 C42B 2110C6                 LXI   H,:C610    Addr INT(10)
539 C42E E7                     RST   4          MACC = MACC * 10 (INT)
540 C42F 54                     DATA  :54
541 C430 E7                     RST   4          Copy MACC to reg A,B,C,D
542 C431 15                     DATA  :15
543 C432 C630                   ADI   :30        Make highest byte ASCII
544 C434 E1                     POP   H
545 C435 C1                     POP   B
546 C436 C9                     RET
547                   *
548                   *
549                   *
550 C437                        END
```

```
*****************************
* S Y M B O L   T A B L E *
*****************************
```

FBC	C361	FCB	C249	FDCM	C1FB	FPM1	C21A
I1	C420	LC233	C34D	LC52	C230	LC53	C250
LC54	C26B	LC55	C26E	LC56	C284	LC57	C29F
LC58	C2A2	LC59	C2A6	LC60	C2A9	LC61	C2AE

LC62	C2BA	LC63	C2D5	LC64	C2D9	LC65	C2DC
LC66	C2DE	LC67	C2EB	LC68	C2F7	LC69	C2FE
LC70	C304	LC71	C311	LC72	C316	LC73	C317
LC74	C32B	LC75	C32D	LC76	C32F	LC77	C34B
LC78	C380	LC79	C38D	LC80	C39D	LC81	C3A2
LC82	C3B3	LC83	C3C4	LC84	C3C8	LC85	C3D4
LC86	C3E4	LC88	C402	LC89	C40E	LC90	C424
POP	C234	PUSH	C21E				

```
002                             ORG   :C437
003                     *
004                     *
005                     *
006                     ************************
007                     * FPT NUMBER CONSTANTS *
008                     ************************
009                     *
010                     * For the first 7 numbers, the 1st byte is the
011                     * power of 10 for division.
012                     *
013 C437 00             LC234   DATA  :00
014 C438 02                     DATA  :02       FPT (2^1)/10^0
015 C439 80                     DATA  :80
016 C43A 00                     DATA  :00
017 C43B 00                     DATA  :00
018                     *
019 C43C 00                     DATA  :00
020 C43D 03                     DATA  :03       FPT (2^2)/10^0
021 C43E 80                     DATA  :80
022 C43F 00                     DATA  :00
023 C440 00                     DATA  :00
024                     *
025 C441 01                     DATA  :01
026 C442 01                     DATA  :01       FPT (2^4)/10^1
027 C443 CC                     DATA  :CC
028 C444 CC                     DATA  :CC
029 C445 CD                     DATA  :CD
030                     *
031 C446 02                     DATA  :02
032 C447 02                     DATA  :02       FPT (2^8)/10^2
033 C448 A3                     DATA  :A3
034 C449 D7                     DATA  :D7
035 C44A 0A                     DATA  :0A
036                     *
037 C44B 04                     DATA  :04
038 C44C 03                     DATA  :03       FPT (2^16)/10^4
039 C44D D1                     DATA  :D1
040 C44E B7                     DATA  :B7
041 C44F 17                     DATA  :17
042                     *
043 C450 09                     DATA  :09
044 C451 03                     DATA  :03       FPT (2^32)/10^9
045 C452 89                     DATA  :89
046 C453 70                     DATA  :70
047 C454 5F                     DATA  :5F
048                     *
049 C455 13                     DATA  :13
050 C456 01                     DATA  :01       FPT (2^64)/10^19
051 C457 EC                     DATA  :EC
052 C458 1E                     DATA  :1E
053 C459 4A                     DATA  :4A
054                     *
055 C45A 7D             LC238   DATA  :7D       FPT (0.1)
056 C45B CC                     DATA  :CC
057 C45C CC                     DATA  :CC
058 C45D CD                     DATA  :CD
059                     *
060 C45E 00             FP0     DATA  :00       FPT (0)
061 C45F 00                     DATA  :00
062 C460 00                     DATA  :00
063 C461 00                     DATA  :00
```

```
064                     *
065 C462 01             FP1     DATA  :01       FPT (1)
066 C463 80                     DATA  :80
067 C464 00                     DATA  :00
068 C465 00                     DATA  :00
069                     *
070 C466 02             FP2     DATA  :02       FPT (2)
071 C467 80                     DATA  :80
072 C468 00                     DATA  :00
073 C469 00                     DATA  :00
074                     *
075 C46A 02             FP3     DATA  :02       FPT (3)
076 C46B C0                     DATA  :C0
077 C46C 00                     DATA  :00
078 C46D 00                     DATA  :00
079                     *
080 C46E 03             FP4     DATA  :03       FPT (4)
081 C46F 80                     DATA  :80
082 C470 00                     DATA  :00
083 C471 00                     DATA  :00
084                     *
085 C472 03             FP5     DATA  :03       FPT (5)
086 C473 A0                     DATA  :A0
087 C474 00                     DATA  :00
088 C475 00                     DATA  :00
089                     *
090 C476 03             FP6     DATA  :03       FPT (6)
091 C477 C0                     DATA  :C0
092 C478 00                     DATA  :00
093 C479 00                     DATA  :00
094                     *
095 C47A 03             FP7     DATA  :03       FPT (7)
096 C47B E0                     DATA  :E0
097 C47C 00                     DATA  :00
098 C47D 00                     DATA  :00
099                     *
100 C47E 04             FP8     DATA  :04       FPT (8)
101 C47F 80                     DATA  :80
102 C480 00                     DATA  :00
103 C481 00                     DATA  :00
104                     *
105 C482 04             FP9     DATA  :04       FPT (9)
106 C483 90                     DATA  :90
107 C484 00                     DATA  :00
108 C485 00                     DATA  :00
109                     *
110                     ************************************
111                     * PRETTIES UP FPT OR INT NUMBER *
112                     ************************************
113                     *
114                     * Entry: B:     Fix/float flag: (0=fix, 1=float).
115                     *        A:     Nr. of useable digits in string in
116                     *               00E4-F0 (not counting additional
117                     *               digit for rounding).
118                     *        00F1:  Nr. of digits before '.' (exponent).
119                     *        00E4:  Sign '+' or '-'.
120                     *        00E5:  Decimal point.
121                     *        E6-F0: Digits.
122                     * Exit:  All registers preserved.
123                     *        00E3:  Length of string.
124                     *        E4-F0: Output string.
125                     *
```

```
126                     * Format: Sign in 00E4 is blank or '-'.
127                     *         If exponent is 0:
128                     *              real case: '0.digits'
129                     *              int. case: no final '.'
130                     *              real case if INT: '.0'
131                     *         If exponent < -1:  E-format
132                     *         If exponent too large: E-format
133                     *         In E-format no '.0'
134                     *
135 C486 F5             PRTY    PUSH  PSW
136 C487 C5                     PUSH  B
137 C488 D5                     PUSH  D
138 C489 E5                     PUSH  H
139 C48A 21E600                 LXI   H,:00E6   Startaddr digits
140 C48D 4F                     MOV   C,A       Save nr. useable digits
141 C48E C5                     PUSH  B
142 C48F 0600                   MVI   B,:00
143 C491 09                     DAD   B         HL pnt to last useable digit
144 C492 7E                     MOV   A,M       Get last digit
145 C493 FE35                   CPI   :35       Check for rounding
146 C495 DAAFC4                 JC    :C4AF     If <5
147 C498 2B             LC91    DCX   H
148 C499 7E                     MOV   A,M       Get digit before
149 C49A FE39                   CPI   :39
150 C49C CAA3C4                 JZ    :C4A3     If it is 9
151 C49F 34                     INR   M         Rounding upwards
152 C4A0 C3AFC4                 JMP   :C4AF     Abort rounding
153 C4A3 3630           LC92    MVI   M,:30     Make digit before 0
154 C4A5 0D                     DCR   C         Decr. nr of digits
155 C4A6 C298C4                 JNZ   :C498     Cont. check for rounding
156 C4A9 3631                   MVI   M,:31     Make nr=1 if all digits 9
157 C4AB 21F100                 LXI   H,:00F1
158 C4AE 34                     INR   M         Incr nr of digits before '.'
159 C4AF C1             LC93    POP   B
160 C4B0 0C                     INR   C         Incr. nr useable digits
161 C4B1 3AF100                 LDA   :00F1     Get nr of digits before '.'
162 C4B4 B7                     ORA   A
163 C4B5 CAD8C4                 JZ    :C4D8     If 0
164 C4B8 FAE1C4                 JM    :C4E1     If too many digits
165 C4BB 80                     ADD   B         Add fix/float flag
166 C4BC B9                     CMP   C
167 C4BD D2E1C4                 JNC   :C4E1     If too many digits
168 C4C0 CD9CC6                 CALL  :C69C     Restore A, insert '.'
169                                             after number string
170 C4C3 79             LC94    MOV   A,C       Length string in A
171 C4C4 CD4BC5                 CALL  :C54B     Calc nr of digits for output
172 C4C7 3C             LC95    INR   A         Add 1
173 C4C8 21E300                 LXI   H,:00E3
174 C4CB 77                     MOV   M,A       String length in outbuf
175 C4CC 23                     INX   H
176 C4CD 7E                     MOV   A,M       Get sign
177 C4CE FE2B                   CPI   :2B       '+' ?
178 C4D0 C2D5C4                 JNZ   :C4D5     Then abort
179 C4D3 3620                   MVI   M,:20     Replace '+' by blank
180 C4D5 C34DC1         LC96    JMP   :C14D     Popall, ret
181
182                     * If format '0.digits':
183
184 C4D8 CD1AC5           LC97    CALL  :C51A     Move string right 1 pos.
185 C4DB 3630                   MVI   M,:30     Insert 0 in 00E5
186 C4DD 0C                     INR   C         Update nr of digits
187 C4DE C3C3C4                 JMP   :C4C3     Update string
```

```
188
189                     * If too many digits:
190
191 C4E1 3E01        LC98    MVI   A,:01
192 C4E3 CD31C5             CALL  :C531      Move string left 1 pos,
193                                           Insert '.' after string
194 C4E6 79                 MOV   A,C        Get nr of digits
195 C4E7 0600               MVI   B,:00
196 C4E9 CD4BC5             CALL  :C54B      Calc nr of digits for output
197 C4EC 47                 MOV   B,A        in B
198 C4ED 3AF100             LDA   :00F1      Get nr of digits before '.'
199 C4F0 3D                 DCR   A          Minus 1
200 C4F1 3645               MVI   M,:45      'E' in buf after last digit
201 C4F3 23                 INX   H
202 C4F4 04                 INR   B          Incr. nr of digits
203 C4F5 B7                 ORA   A          Flags on exponent
204 C4F6 F2FFC4             JP    :C4FF      If exp. positive
205
206                     * If exponent negative:
207
208 C4F9 362D               MVI   M,:2D      Store '-' in buffer
209 C4FB 23                 INX   H
210 C4FC 04                 INR   B          Incr nr of digits
211 C4FD 2F                 CMA
212 C4FE 3C                 INR   A          2-compl of exponent
213
214                     * Exponent to buffer:
215
216 C4FF 110A2F      LC99    LXI   D,:2F0A
217 C502 93          LC100   SUB   E          Exp.-10 (unit value)
218 C503 14                 INR   D          ASCII-count 10's-value
219 C504 D202C5             JNC   :C502      If rest exp. still >10
220 C507 C63A               ADI   :3A        Convert rest to Ascii
221 C509 5F                 MOV   E,A        in E
222 C50A 7A                 MOV   A,D        Get 10's value
223 C50B FE30               CPI   :30
224 C50D CA13C5             JZ    :C513      If exp. <10
225 C510 77                 MOV   M,A        10's value exp. in buf
226 C511 23                 INX   H
227 C512 04                 INR   B          Incr nr of digits
228 C513 73          LC101   MOV   M,E        Unit value exp. in buf
229 C514 23                 INX   H
230 C515 04                 INR   B          Incr nr of digits
231 C516 78                 MOV   A,B        into A
232 C517 C3C7C4             JMP   :C4C7      Prepare string for output
233                     *
234                     * MOVE STRING IN OUTPUTBUFFER RIGHT 1 POS.:
235                     *
236                     * The contents of 00E5-EF is moved up one
237                     * position to 00E6-F0.
238                     *
239                     * Entry: No conditions.
240                     * Exit:  ABCDEF preserved.
241                     *        HL points to 00E5.
242                     *
243 C51A F5          LC102   PUSH  PSW
244 C51B C5                 PUSH  B
245 C51C D5                 PUSH  D
246 C51D 21F000             LXI   H,:00F0    Highest destination address.
247 C520 54                 MOV   D,H
248 C521 5D                 MOV   E,L
249 C522 1B                 DCX   D          Highest source address.
```

```
C523 060B            MVI   B,:0B     Number of bytes.
C525 1A      LC103   LDAX  D         Get byte
C526 77              MOV   M,A       and move it.
C527 1B              DCX   D
C528 2B              DCX   H
C529 05              DCR   B
C52A C225C5          JNZ   :C525     Next byte if not ready
C52D D1              POP   D
C52E C1              POP   B
C52F F1              POP   PSW
C530 C9              RET
                *
                * MOVE STRING IN OUTPUTBUFFER LEFT 1 POS.:
                *
                * The string, beginning on 00E6, is moved one
                * memory location downwards. A '.' is inserted
                * after the string.
                *
                * Entry: A: number of bytes to be transferred.
                * Exit:  All registers preserved.
                *
C531 F5      LC104   PUSH  PSW
C532 C5              PUSH  B
C533 D5              PUSH  D
C534 E5              PUSH  H
C535 47              MOV   B,A       Store number of bytes
C536 21E500          LXI   H,:00E5   Lowest destination address
C539 11E600          LXI   D,:00E6   Lowest source address
C53C 1A      LC105   LDAX  D         Get byte
C53D 77              MOV   M,A       and move it
C53E 13              INX   D
C53F 23              INX   H
C540 05              DCR   B
C541 C23CC5          JNZ   :C53C     Next byte if not ready
C544 362E            MVI   M,:2E     Insert '.' after string
C546 E1              POP   H
C547 D1              POP   D
C548 C1              POP   B
C549 F1              POP   PSW
C54A C9              RET
                *
                * CALCULATE NUMBER OF DIGITS FOR OUTPUT:
                *
                * Entry: Total nr of string digits in A and C.
                *        B: Flag for INT (0) or FPT (1).
                *        Digits in 00E4 to 00E4 + A.
                * Exit:  A:  Nr of bytes for output:
                *                 INT: excl. trailing '.0'
                *                 FPT: incl. trailing '.0'
                *        HL: If last non-zero byte is not '.':
                *            points after last byte.
                *            Else: INT: points to '.'
                *                  FPT: after '.0'
                *
C54B C5      LC106   PUSH  B
C54C D5              PUSH  D
C54D 21E400          LXI   H,:00E4   Startaddr string
C550 5F              MOV   E,A       Total nr of digits in E
C551 1600            MVI   D,:00
C553 19              DAD   D         Calc end of string
C554 7E      LC107   MOV   A,M       Get digit
C555 FE30            CPI   :30
```

```
C557 C25FC5             JNZ   :C55F     If non-zero
C55A 2B                 DCX   H         Points to previous digit
C55B 0D                 DCR   C         Decr nr of digits
C55C C354C5             JMP   :C554     Again till non-zero found

                * If non-zero digit found:

C55F FE2E       LC108   CPI   :2E       '.' ?
C561 23                 INX   H
C562 C26FC5             JNZ   :C56F     Abort if not
C565 2B                 DCX   H         Pnts after last non-zero,
                                        non-'.' digit
C566 0D                 DCR   C         Excl. '.'
C567 05                 DCR   B
C568 C26FC5             JNZ   :C56F     If INT case
C56B 23                 INX   H         ) If FPT case: pnts
C56C 23                 INX   H         ) after '.0'
C56D 0C                 INR   C
C56E 0C                 INR   C         Incl. '.0'
C56F 79         LC109   MOV   A,C       Nr of digits for output
C570 D1                 POP   D
C571 C1                 POP   B
C572 C9                 RET
                *
                ********************************
                * INPUT INTEGER NUMBER TO MACC *
                ********************************
                *
                * Read string of digits from line and convert
                * it to binary in MACC.
                *
                * Entry: BC points to input character.
                * Exit:  BC points after INT number.
                *        ADEHL preserved.
                *        CY=1: there were digits.
                *        CY=0: No digits.
                *
C573 37         ICB     STC
C574 F5                 PUSH  PSW
C575 D5                 PUSH  D
C576 E5                 PUSH  H
C577 CD98C5             CALL  :C598     Clear MACC and 00E3-E6.
C57A CD73C0     LC110   CALL  :C073     Get digit from line
C57D D630               SUI   :30       Convert ASCII to binary
C57F DA90C5             JC    :C590     )
C582 FE0A               CPI   :0A       ) Abort if no number
C584 D290C5             JNC   :C590     )
C587 2110C6             LXI   H,:C610   Addr INT(10)
C58A CDA5C5             CALL  :C5A5     MACC=MACC*10 + digit
C58D C37AC5             JMP   :C57A     Next digit
C590 15         LC111   DCR   D
C591 14                 INR   D
C592 C2A2C2             JNZ   :C2A2     If digits: Pop, ret
C595 C3A9C2             JMP   :C2A9     If no digits: CY=0, Pop, ret
                *
                * CLEAR MACC AND 00E3-00E6:
                *
                * Both MACC and registers 00E3-E6 are loaded
                * with the value of FPT (0).
                *
                * Exit: ABCE preserved. D=0.
                *
```

```
374 C598 215EC4     LC112   LXI   H,:C45E   Addr. FPT(0)
375 C59B E7                 RST   4         Copy FPT(0) to MACC
376 C59C 0C                 DATA  :0C
377 C59D 21E300             LXI   H,:00E3
378 C5A0 E7                 RST   4         Copy FPT(0) to 00E3-E6
379 C5A1 0F                 DATA  :0F
380 C5A2 1600               MVI   D,:00     Clear D
381 C5A4 C9                 RET
382                 *
383                 * MACC = MACC*10 + DIGIT FROM LINE:
384                 *
385                 * Entry: HL: points to INT (10).
386                 *         A:  digit to be added.
387                 *
388 C5A5 E7         LC113   RST   4         MACC=MACC*10 (INT)
389 C5A6 54                 DATA  :54
390 C5A7 0C         LC114   INR   C
391 C5A8 15                 DCR   D
392 C5A9 32E600             STA   :00E6     Digit in lobyte E3-E6
393 C5AC 21E300             LXI   H,:00E3
394 C5AF E7                 RST   4         Add (E3-E6) to MACC (INT)
395 C5B0 4E                 DATA  :4E
396 C5B1 C9                 RET
397                 *
398                 ****************************************
399                 * CONVERT INTEGER NUMBER FOR OUTPUT *
400                 ****************************************
401                 *
402                 * Places ASCII string from INT MACC contents in
403                 * output buffer 00E4-F0.
404                 * 00E4 is sign, 00E5 is '.', 00E6-F0 is value,
405                 * 00F1 is nr of digits.
406                 *
407                 * Exit: A: Number of digits.
408                 *       BCDEHL preserved.
409                 *
410 C5B2 C5         IBC     PUSH  B
411 C5B3 D5                 PUSH  D
412 C5B4 E5                 PUSH  H
413 C5B5 CD1EC2             CALL  :C21E     Save MACC to TOS
414 C5B8 CDE0C5             CALL  :C5E0     Abs.value of MACC in reg
415                                          A,B,C,D; Prepare 00E4-E6
416 C5BB CD1EC2     LC115   CALL  :C21E     Save MACC to TOS
417 C5BE 2110C6             LXI   H,:C610   Addr INT(10)
418 C5C1 E7                 RST   4         MACC = remainder MACC/10
419 C5C2 5A                 DATA  :5A
420 C5C3 E7                 RST   4         Copy MACC to reg A,B,C,D
421 C5C4 15                 DATA  :15
422 C5C5 7A                 MOV   A,D       Lobyte in A
423 C5C6 CDFAC5             CALL  :C5FA     Digit into 00E5-F0
424 C5C9 CD34C2             CALL  :C234     Retrieve MACC from TOS
425 C5CC E7                 RST   4         MACC = MACC/10 (INT)
426 C5CD 57                 DATA  :57
427 C5CE E7                 RST   4         Copy MACC to reg A,B,C,D
428 C5CF 15                 DATA  :15
429 C5D0 B0                 ORA   B
430 C5D1 B1                 ORA   C
431 C5D2 B2                 ORA   D
432 C5D3 C2BBC5             JNZ   :C5BB     Again if <>0
433 C5D6 CD06C6             CALL  :C606     '.' in 00E5; length in 00F1
434 C5D9 CD34C2             CALL  :C234     Retrieve MACC from TOS
435 C5DC E1                 POP   H
```

```
C5DD D1                 POP   D
C5DE C1                 POP   B
C5DF C9                 RET
                *
                * PREPARE 00E4-E6:
                *
                * 00E4-E6 is set to +00 or -00, depending on sign
                * of contents MACC. In the MACC remains the absolute
                * value. The registers A,B,C,D contain the original
                * contents of the MACC.
                *
                * Exit: E=0. HL preserved. AFBCD corrupted.
                *
C5E0 E5         LC116   PUSH  H
C5E1 21E400             LXI   H,:00E4
C5E4 E7                 RST   4           Copy MACC to reg A,B,C,D
C5E5 15                 DATA  :15
C5E6 B7                 ORA   A           Set flags on sign
C5E7 362B               MVI   M,:2B       '+' in 00E4
C5E9 F2F0C5             JP    :C5F0       Jump if nr is positive
C5EC 362D               MVI   M,:2D       Else '-' in 00E4
C5EE E7                 RST   4           and make contents MACC pos.
C5EF 60                 DATA  :60
C5F0 23         LC117   INX   H
C5F1 3630               MVI   M,:30       0 in 00E5
C5F3 23                 INX   H
C5F4 3630               MVI   M,:30       0 in 00E6
C5F6 1E00               MVI   E,:00       Digit count is 0
C5F8 E1                 POP   H
C5F9 C9                 RET
                *
                * STORE DIGIT IN OUTPUT BUFFER 00E5-00F0:
                *
                * Entry: Digit in A.
                * Exit:  Digit in 00E5-F0 as most sign. digit.
                *        E:  Count of digit in buffer.
                *        BCDHL preserved. AF corrupted.
                *
C5FA E5         LC118   PUSH  H
C5FB F5                 PUSH  PSW
C5FC CD1AC5             CALL  :C51A       Move contents buffer right
C5FF F1                 POP   PSW
C600 C630               ADI   :30         Make digit ASCII
C602 77                 MOV   M,A         Digit in 00E5 inserted.
C603 1C                 INR   E           Update digit count.
C604 E1                 POP   H
C605 C9                 RET
                *
                * ADD A '.' TO A DIGIT STRING IN OUTPUTBUFFER:
                *
                * A '.' is placed at the beginning of a digit
                * string in the output buffer. The length of the
                * string is stored in 00F1.
                *
                * Entry: E:  Digit count.
                *        HL: Points to 00E5.
                * Exit:  A:  Count.
                *        BCDEHL preserved.
                *
C606 CD1AC5     LC119   CALL  :C51A       Move contents outbuf right
                                          one position
C609 362E               MVI   M,:2E       '.' at begin of string
```

```
498 C60B 7B                     MOV   A,E       Get digit count
499 C60C 32F100                 STA   :00F1     Store it in buffer
500 C60F C9                     RET
501                   *
502                   * DATA:
503                   *
504 C610 00           I10       DATA  :00       INT (10)
505 C611 00                     DATA  :00
506 C612 00                     DATA  :00
507 C613 0A                     DATA  :0A
508                   *
509                   *
510                   *
511 C614                        END
```

****
* SYMBOL TABLE *
****

FP0	C45E	FP1	C462	FP2	C466	FP3	C46A
FP4	C46E	FP5	C472	FP6	C476	FP7	C47A
FP8	C47E	FP9	C482	I10	C610	IBC	C5B2
ICB	C573	LC100	C502	LC101	C513	LC102	C51A
LC103	C525	LC104	C531	LC105	C53C	LC106	C54B
LC107	C554	LC108	C55F	LC109	C56F	LC110	C57A
LC111	C590	LC112	C598	LC113	C5A5	LC114	C5A7
LC115	C5BB	LC116	C5E0	LC117	C5F0	LC118	C5FA
LC119	C606	LC234	C437	LC238	C45A	LC91	C498
LC92	C4A3	LC93	C4AF	LC94	C4C3	LC95	C4C7
LC96	C4D5	LC97	C4D8	LC98	C4E1	LC99	C4FF
PRTY	C486						

```
002                             ORG   :C614
003                     *
004                     *
005                     *
006                     ******************************
007                     * INPUT HEX NUMBER TO MACC *
008                     ******************************
009                     *
010                     * Reads a sequence of hex digits and converts
011                     * them into MACC.
012                     *
013                     * Entry: C points to input.
014                     * Exit:  CY=1: There was a digit.
015                     *        CY=0: No digit.
016                     *        C points to next input.
017                     *        ABDEHL preserved.
018                     *
019 C614 37             HCB     STC
020 C615 F5                     PUSH  PSW
021 C616 D5                     PUSH  D
022 C617 E5                     PUSH  H
023 C618 CD98C5                 CALL  :C598     Clear MACC and 00E3-E6
024 C61B CD73C0         LC120   CALL  :C073     Get digit from line
025 C61E D630                   SUI   :30       )
026 C620 DA90C5                 JC    :C590     )
027 C623 FE0A                   CPI   :0A       ) Check if hex number
028 C625 DA34C6                 JC    :C634     )
029 C628 D607                   SUI   :07       ) Abort via C590 if not
030 C62A FE0A                   CPI   :0A       )
031 C62C DA90C5                 JC    :C590     )
032 C62F FE10                   CPI   :10       )
033 C631 D290C5                 JNC   :C590     )
034 C634 213DC6         LC121   LXI   H,:C63D   Addr INT(4)
035 C637 CD41C6                 CALL  :C641     Insert digit at low end
036                                             MACC
037 C63A C31BC6                 JMP   :C61B     Get next digit
038                     *
039                     * DATA:
040                     *
041 C63D 00             I4      DATA  :00       INT (4)
042 C63E 00                     DATA  :00
043 C63F 00                     DATA  :00
044 C640 04                     DATA  :04
045                     *
046                     * ENTER HEX DIGIT AT LOW END MACC:
047                     *
048                     * Entry: HL points to a 4-byte number.
049                     *        A  contains a digit.
050                     * Exit:  HL = 00E3.
051                     *        C is incremented, D decremented.
052                     *        ABE preserved.
053                     *
054 C641 F5             LC122   PUSH  PSW
055 C642 C5                     PUSH  B
056 C643 D5                     PUSH  D
057 C644 E7                     RST   4         Copy MACC to reg A,B,C,D
058 C645 15                     DATA  :15
059 C646 E6F0                   ANI   :F0       Check if value too high
060 C648 C44BC0                 CNZ   :C04B     Then overflow error
061 C64B D1                     POP   D
062 C64C C1                     POP   B
063 C64D F1                     POP   PSW
```

```
064 C64E E7                     RST   4          Shift left
065 C64F 6F                     DATA  :6F
066 C650 C3A7C5                 JMP   :C5A7      Add digit to MACC
067                     *
068                     ****************************************
069                     * CONVERT HEX MACC TO ASCII FOR OUTPUT *
070                     ****************************************
071                     *
072                     * Converts a HEX number in MACC into its ASCII
073                     * representation into the output buffer.
074                     * Not significant leading zeroes are cancelled.
075                     *
076                     * Exit: BCDEHL preserved. AF corrupted.
077                     *       Length output string in 00E3.
078                     *       Output string in 00E4-00EB.
079                     *
080 C653 C5             HBC     PUSH  B
081 C654 D5                     PUSH  D
082 C655 E5                     PUSH  H
083 C656 CD8DC6                 CALL  :C68D      Get startaddr DECBUF in HL
084 C659 E7                     RST   4          Copy MACC to reg A,B,C,D
085 C65A 15                     DATA  :15
086 C65B CD6AC6                 CALL  :C66A      Convert A,B to ASCII
087                                              into DECBUF
088 C65E 79                     MOV   A,C
089 C65F 42                     MOV   B,D
090 C660 CD6AC6                 CALL  :C66A      Idem for C,D
091 C663 CD91C6                 CALL  :C691      Get string length in 00E3
092 C666 E1                     POP   H
093 C667 D1                     POP   D
094 C668 C1                     POP   B
095 C669 C9                     RET
096
097                     * Convert 2 hex digits:
098
099 C66A CD6EC6         LC123   CALL  :C66E      Convert 1st digit
100 C66D 78                     MOV   A,B        Get 2nd one
101 C66E F5             LC124   PUSH  PSW
102 C66F 1F                     RAR
103 C670 1F                     RAR
104 C671 1F                     RAR
105 C672 1F                     RAR              Shift high nibble in low
106 C673 CD77C6                 CALL  :C677      Convert it to ASCII
107 C676 F1                     POP   PSW        Restore both nibbles
108 C677 E60F           LC125   ANI   :0F        Low nibble only
109 C679 FE0A                   CPI   :0A
110 C67B DA80C6                 JC    :C680      If 0 < digit < 9
111 C67E C607                   ADI   :07        Add 7 for A < digit < F
112 C680 C630           LC126   ADI   :30        Convert to ASCII
113 C682 77                     MOV   M,A        Into DECBUF
114 C683 23                     INX   H          Incr pointer
115 C684 FE30                   CPI   :30
116 C686 C0                     RNZ              Abort if digit <> 0
117
118                     * If 1st digit is zero:
119
120 C687 7D                     MOV   A,L        Get lsbyte buffer pointer
121 C688 FEE5                   CPI   :E5        1st digit in buffer?
122 C68A C0                     RNZ              Abort if not
123 C68B 2B                     DCX   H          Else: cancel non-sign. 0's
124 C68C C9                     RET
125
```

```
                        * Get startaddress output buffer:
                        *
C68D 21E400     LC127   LXI   H,:00E4   Startaddr in HL
C690 C9                 RET
                        *
                        * CALCULATE LENGTH OF STRING IN OUTPUT BUFFER:
                        *
                        * Entry: L: lobyte of address last digit in buffer.
                        * Exit:  BCDEHL preserved. AF corrupted.
                        *        Length is stored in 00E3.
                        *
C691 7D         LC128   MOV   A,L       Get lobyte addr last digit
C692 D6E4               SUI   :E4       Minus beginaddr
C694 FE01               CPI   :01
C696 CE00               ACI   :00       Length min. 1
C698 32E300             STA   :00E3     Store length in DECBUF
C69B C9                 RET
                        *
                        ******************************************
                        * RESTORE A, ADD '.' AFTER DIGIT STRING *
                        ******************************************
                        *
                        * Part of PRTY (C486).
                        *
C69C 90         LC129   SUB   B         Restore A
C69D C331C5             JMP   :C531     Move string left 1 pos,
                                        insert '.'
                        *
                        ****************************************
                        * PRINT CHARACTER, INPUT A TEXT LINE *
                        ****************************************
                        *
                        * Part of Run 'INPUT' (0E3D6).
                        *
                        * Entry: Character in A.
                        * Exit:  BC preserved.
                        *
C6A0 C5         PINPLN  PUSH  B
C6A1 CD1FDD             CALL  :DD1F     Print char; input textline
C6A4 C1                 POP   B
C6A5 C9                 RET
                        *
C6A6 FF                 DATA  :FF
C6A7 FF                 DATA  :FF
                        *
                        **********************
                        * DATA FOR 'RANDOM' *
                        **********************
                        *
C6A8 00         RNDA    DATA  :00       Random number constant A
C6A9 00                 DATA  :00
C6AA 00                 DATA  :00
C6AB 3B                 DATA  :3B
                        *
C6AC 07         RNDB    DATA  :07       Random number constant B
C6AD 73                 DATA  :73
C6AE 59                 DATA  :59
C6AF 41                 DATA  :41
                        *
C6B0 01         IROR    DATA  :01       OR mask  (FPT (1))
C6B1 80                 DATA  :80
C6B2 00                 DATA  :00
```

```
188 C6B3 00                 DATA  :00
189                 *
190                 ******************************
191                 * part of READ BLOCK (D340) *
192                 ******************************
193                 *
194                 * Exit if no loading errors.
195                 *
196 C6B4 E3         MPT26   XTHL
197 C6B5 37                 STC             CY=1: no error
198 C6B6 E1         LBK30   POP   H
199 C6B7 D1                 POP   D
200 C6B8 C1                 POP   B
201 C6B9 C9                 RET
202                 *
203                 ******************
204                 * part of 2E8DE *
205                 ******************
206                 *
207 C6BA CD91CE     SPT02   CALL  :CE91     Go and set screen bits for
208                                         mode 1
209 C6BD C338E1             JMP   :E138     (2) Pop all, ret.
210                 *
211                 *
212                 *  ================
213                 *** BANK SWITCHING ***
214                 *  ================
215                 *
216                 *
217                 ***************************
218                 * MATH. RESTART (RST 4) *
219                 ***************************
220                 *
221                 * This, and the following routines, switch the
222                 * paged banks of ROM. They are entered via
223                 * RST x; DATA xx instructions.
224                 *
225 C6C0 E1         MARST   POP   H
226 C6C1 F3                 DI
227 C6C2 224300             SHLD  :0043     Save HL
228 C6C5 F5                 PUSH  PSW
229 C6C6 E1                 POP   H
230 C6C7 224100             SHLD  :0041     Save PSW
231 C6CA 2640               MVI   H,:40     ROM bank 1 select bits
232 C6CC 3AD400             LDA   :00D4     Offset of start HW/SW vector
233                 *
234                 * ROM BANK SWITCHING:
235                 *
236                 * This routine is generally used by all Restarts
237                 * using ROM bank switching.
238                 *
239 C6CF E3         MRS10   XTHL
240 C6D0 86                 ADD   M         Add entry number
241 C6D1 23                 INX   H
242 C6D2 E3                 XTHL
243 C6D3 6F                 MOV   L,A       Complete entry point address
244 C6D4 3A4000             LDA   :0040     Old bank select port status
245 C6D7 F5                 PUSH  PSW       Save it
246 C6D8 E63F               ANI   :3F       Keep other bits
247 C6DA B4                 ORA   H         Add new select bits
248 C6DB 324000             STA   :0040     Update memory
249 C6DE 3206FD             STA   :FD06     Update port
```

```
188 C6B3 00                 DATA  :00
189                 *
190                 ******************************
191                 * part of READ BLOCK (D340) *
192                 ******************************
193                 *
194                 * Exit if no loading errors.
195                 *
196 C6B4 E3         MPT26   XTHL
197 C6B5 37                 STC             CY=1: no error
198 C6B6 E1         LBK30   POP   H
199 C6B7 D1                 POP   D
200 C6B8 C1                 POP   B
201 C6B9 C9                 RET
202                 *
203                 ******************
204                 * part of 2E8DE *
205                 ******************
206                 *
207 C6BA CD91CE     SPT02   CALL  :CE91     Go and set screen bits for
208                                         mode 1
209 C6BD C338E1             JMP   :E138     (2) Pop all, ret.
210                 *
211                 *
212                 *  ================
213                 *** BANK SWITCHING ***
214                 *  ================
215                 *
216                 *
217                 ***************************
218                 * MATH. RESTART (RST 4) *
219                 ***************************
220                 *
221                 * This, and the following routines, switch the
222                 * paged banks of ROM. They are entered via
223                 * RST x; DATA xx instructions.
224                 *
225 C6C0 E1         MARST   POP   H
226 C6C1 F3                 DI
227 C6C2 224300             SHLD  :0043     Save HL
228 C6C5 F5                 PUSH  PSW
229 C6C6 E1                 POP   H
230 C6C7 224100             SHLD  :0041     Save PSW
231 C6CA 2640               MVI   H,:40     ROM bank 1 select bits
232 C6CC 3AD400             LDA   :00D4     Offset of start HW/SW vector
233                 *
234                 * ROM BANK SWITCHING:
235                 *
236                 * This routine is generally used by all Restarts
237                 * using ROM bank switching.
238                 *
239 C6CF E3         MRS10   XTHL
240 C6D0 86                 ADD   M         Add entry number
241 C6D1 23                 INX   H
242 C6D2 E3                 XTHL
243 C6D3 6F                 MOV   L,A       Complete entry point address
244 C6D4 3A4000             LDA   :0040     Old bank select port status
245 C6D7 F5                 PUSH  PSW       Save it
246 C6D8 E63F               ANI   :3F       Keep other bits
247 C6DA B4                 ORA   H         Add new select bits
248 C6DB 324000             STA   :0040     Update memory
249 C6DE 3206FD             STA   :FD06     Update port
```

```
250 C6E1 26E0              MVI   H,:E0
251 C6E3 CDF2C6            CALL  :C6F2     Restore HL, PSW; switch bank
252
253                  * Return from switched bank:
254
255 C6E6 E3                XTHL            Return to old bank
256 C6E7 F5                PUSH  PSW
257 C6E8 7C                MOV   A,H       Get old bank
258 C6E9 324000            STA   :0040     Reinstate memory
259 C6EC 3206FD            STA   :FD06     Re-instate port
260 C6EF F1                POP   PSW
261 C6F0 E1                POP   H
262 C6F1 C9                RET             Return to caller
263                  *
264                  * SWITCH TO ROM BANK:
265                  *
266                  * HL and PSW are restored. On exit, the
267                  * program switches to the selected ROM bank.
268                  * After return from this bank, the program
269                  * continues on C6E6.
270                  *
271 C6F2 E5          MRDCL   PUSH  H
272 C6F3 2A4100              LHLD  :0041
273 C6F6 E5                PUSH  H
274 C6F7 F1                POP   PSW       Restore A,F
275 C6F8 2A4300            LHLD  :0043     Restore H,L
276 C6FB FB                EI
277 C6FC C9                RET             Switch to selected bank
278                  *
279                  ***************************
280                  * SCREEN RESTART (RST 5) *
281                  ***************************
282                  *
283                  * From SRS10 also used by RST 1.
284                  *
285 C6FD E1          SCRST   POP   H
286 C6FE F3                DI
287 C6FF 224300            SHLD  :0043     Save HL
288 C702 F5                PUSH  PSW
289 C703 3E80              MVI   A,:80     ROM bank 2 select bits
290 C705 E1          SRS10   POP   H
291 C706 224100            SHLD  :0041     Save PSW
292 C709 67                MOV   H,A
293 C70A AF                XRA   A
294 C70B C3CFC6            JMP   :C6CF     Switch to new bank
295                  *
296                  ***************************
297                  * UTILITY/ENCODE (RST 1) *
298                  ***************************
299                  *
300 C70E E1          UTRST   POP   H
301 C70F F3                DI
302 C710 224300            SHLD  :0043     Save HL
303 C713 F5                PUSH  PSW
304 C714 3EC0              MVI   A,:C0     ROM bank 3 select bits
305 C716 C305C7            JMP   :C705     Switch to ROM bank 3
306                  *
307                  *
308                  *  ==============
309                  *** BASIC HANDLER ***
310                  *  ==============
311                  *
```

```
312                     *
313                     *********
314                     * RESET *
315                     *********
316                     *
317                     * BASIC entry point. Entry via hardware reset by
318                     * means of a bootstrap on the addresslines to C000.
319                     *
320                     * This section is responsible for all 'once only'
321                     * initialisation of the hardware and the software
322                     * environment. It initialises pointers to all RAM
323                     * areas required, the interrupt system and the
324                     * software modules.
325                     *
326                     INIT
327 C719 3100F9         RESET   LXI   SP,:F900  Init. stackpointer
328 C71C 3E30                 MVI   A,:30     ) cassette motors off;
329 C71E 324000               STA   :0040     ) paddle enable off;
330 C721 3206FD               STA   :FD06     ) select ROM bank 0
331 C724 CDFBD8               CALL  :D8FB     Init. interrupt system
332 C727 210000               LXI   H,:0000
333 C72A 22A302               SHLD  :02A3     Set for no Basic
334 C72D AF                   XRA   A
335 C72E 329302               STA   :0293     RNDLY=0
336 C731 00                   NOP
337 C732 00                   NOP
338
339                     * Init. math.package:
340
341 C733 11F2C7               LXI   D,:C7F2   Addr table error vectors
342 C736 21E0DD               LXI   H,:DDE0   Addr routine get char/line
343 C739 CD03C0               CALL  :C003     Package initialisation
344
345                     * Init. screen RAM:
346
347 C73C CDFBC7               CALL  :C7FB     Check available RAM space
348 C73F 2B                   DCX   H         Highest RAM address
349 C740 11E0C7               LXI   D,:C7E0   Addr screen default data
350 C743 EF                   RST   5
351 C744 00                   DATA  :00       Init. screen RAM
352
353                     * Init. I/O:
354
355 C745 AF                   XRA   A
356 C746 CD8DEE               CALL  :EE8D     (0) Init. I/O switching
357                                           (input keyb; output screen
358                                           + RS232)
359 C749 323501               STA   :0135     Input from keyboard for
360                                           encoding
361 C74C 3EC0                 MVI   A,:C0
362 C74E 32F5FF               STA   :FFF5     Init. TICC baud rate
363
364                     * Init. screen:
365
366 C751 CDFFDA               CALL  :DAFF     Print 'DAI PERSONAL
367 C754 A8C7                 DBL   :C7A8     COMPUTER'
368 C756 2AA502               LHLD  :02A5     Get bottom screen RAM
369 C759 117B09               LXI   D,:097B
370 C75C 19                   DAD   D         Get line mode byte of line
371                                           with 'DAI pC'
372 C75D 365F                 MVI   M,:5F     Set for medium resolution
373 C75F 11D0FF               LXI   D,:FFD0
```

```
374 C762 19                     DAD   D         Create new line mode byte
375                                               for line with 'COMPUTER'
376 C763 CDF9CE                 CALL  :CEF9     Place 'COMPUTER' on new line
377 C766 160F                   MVI   D,:0F     Nr of blanking lines
378 C768 CDCFCE       HD10      CALL  :CECF     Blank next 15 lines
379 C76B 15                     DCR   D
380 C76C C268C7                 JNZ   :C768     Next line
381
382                   * Prepare BASIC:
383
384 C76F CD2DD7                 CALL  :D72D     Init. Soundgen/DCEbus/
385                                               transfer cassette data/
386                                               set start HEAP/get evt.
387                                               DCE-inputs
388 C772 210001                 LXI   H,:0100
389 C775 229D02                 SHLD  :029D     HEAP size default value
390 C778 CDB5DE                 CALL  :DEB5     Run 'NEW'
391 C77B 3E10                   MVI   A,:10
392 C77D 323D01                 STA   :013D     Select cassette port 1
393 C780 21C5E8                 LXI   H,:E8C5   (3) Ptr. to ASCII table
394 C783 CD60D5                 CALL  :D560     Init keyboard pointers
395 C786 CD01D1                 CALL  :D101     Init string handler
396 C789 3E00                   MVI   A,:00
397 C78B 328F02                 STA   :028F     Default number type FPT
398 C78E 117502                 LXI   D,:0275   Begin IMPTAB
399 C791 218F02                 LXI   H,:028F   End IMPTAB
400 C794 CD7CDE                 CALL  :DE7C     Init. implicit type table
401                                               with 0 (= FPT)
402 C797 FB                     EI
403 C798 CF                     RST   1         Wait for input keyboard
404 C799 15                     DATA  :15       or RS232
405 C79A 00                     NOP
406 C79B 21EEC7                 LXI   H,:C7EE   Ptr. to mode 0 colours
407 C79E EF                     RST   5         Set text colours
408 C79F 06                     DATA  :06
409
410                   * Entry from utility:
411
412 C7A0 CDE4CE       RINIT     CALL  :CEE4     Select ROM bank 0 and print
413 C7A3 D3C7                   DBL   :C7D3     'BASIC V1.0'
414 C7A5 C318C8                 JMP   :C818     Into BASIC monitor
415                   *
416                   * INITIALISATION SCREEN DATA:
417                   *
418 C7A8 0D           MSGHDR    DATA  :0D       Frigged screen header
419 C7A9 0D                     DATA  :0D
420 C7AA 0D                     DATA  :0D
421 C7AB 0D                     DATA  :0D
422 C7AC 0D                     DATA  :0D
423 C7AD 0D                     DATA  :0D
424 C7AE 20                     DATA  :20
425 C7AF 44                     DATA  :44       D
426 C7B0 41                     DATA  :41       A
427 C7B1 49                     DATA  :49       I
428 C7B2 20                     DATA  :20
429 C7B3 50                     DATA  :50       P
430 C7B4 45                     DATA  :45       E
431 C7B5 52                     DATA  :52       R
432 C7B6 53                     DATA  :53       S
433 C7B7 4F                     DATA  :4F       O
434 C7B8 4E                     DATA  :4E       N
435 C7B9 41                     DATA  :41       A
```

```
436 C7BA 4C                     DATA  :4C       L
437 C7BB 20                     DATA  :20
438 C7BC 20                     DATA  :20
439 C7BD 20                     DATA  :20
440 C7BE 20                     DATA  :20
441 C7BF 20                     DATA  :20
442 C7C0 20                     DATA  :20
443 C7C1 20                     DATA  :20
444 C7C2 20                     DATA  :20
445 C7C3 20                     DATA  :20
446 C7C4 20                     DATA  :20
447 C7C5 20                     DATA  :20
448 C7C6 20                     DATA  :20
449 C7C7 20                     DATA  :20
450 C7C8 20                     DATA  :20
451 C7C9 43                     DATA  :43       C
452 C7CA 4F                     DATA  :4F       O
453 C7CB 4D                     DATA  :4D       M
454 C7CC 50                     DATA  :50       P
455 C7CD 55                     DATA  :55       U
456 C7CE 54                     DATA  :54       T
457 C7CF 45                     DATA  :45       E
458 C7D0 52                     DATA  :52       R
459 C7D1 0D                     DATA  :0D
460 C7D2 00                     DATA  :00
461                     *
462 C7D3 0C             MSGIN   DATA  :0C
463 C7D4 42                     DATA  :42       B
464 C7D5 41                     DATA  :41       A
465 C7D6 53                     DATA  :53       S
466 C7D7 49                     DATA  :49       I
467 C7D8 43                     DATA  :43       C
468 C7D9 20                     DATA  :20
469 C7DA 56                     DATA  :56       V
470 C7DB 31                     DATA  :31       1
471 C7DC 2E                     DATA  :2E       .
472 C7DD 30                     DATA  :30       0
473 C7DE 0D                     DATA  :0D
474 C7DF 00                     DATA  :00
475                     *
476                     * SCREEN INITIALISATION PARAMETERS:
477                     *
478 C7E0 01             SIPAR   DATA  :01       Default cursor type
479 C7E1 5F                     DATA  :5F       Default cursor ASCII value
480                     *
481 C7E2 05                     DATA  :05       )
482 C7E3 0F                     DATA  :0F       ) Colours COLORT
483 C7E4 0F                     DATA  :0F       ) during Reset
484 C7E5 05                     DATA  :05       )
485                     *
486 C7E6 00                     DATA  :00       )
487 C7E7 05                     DATA  :05       ) Default colours
488 C7E8 0A                     DATA  :0A       ) COLORG
489 C7E9 0F                     DATA  :0F       )
490                     *
491 C7EA 01CA                   DBL   :CA01     Addr. memory management
492                                             routine
493 C7EC 25CA                   DBL   :CA25     Addr. emergency stop routine
494                     *
495 C7EE 08             STCOL   DATA  :08       )
496 C7EF 00                     DATA  :00       ) Default colours
497 C7F0 00                     DATA  :00       ) COLORT
```

```
498 C7F1 08                     DATA  :08        )
499                     *
500                     * MATH. ERROR ROUTINE VECTORS:
501                     *
502 C7F2 1FDA           MEVEC   DBL   :DA1F      Addr. Overflow error routine
503 C7F4 15DA                   DBL   :DA15      Addr. Number out of range
504                                              error routine
505 C7F6 FAC7                   DBL   :C7FA      Addr. Return
506 C7F8 24DA                   DBL   :DA24      Addr. Error routine Division
507                                              by zero
508                     *
509 C7FA C9             MERET   RET              Return
510                     *
511                     *
512                     *
513 C7FB                        END
```

```
*****************************
* S Y M B O L   T A B L E *
*****************************
```

HBC	C653	HCB	C614	HD10	C768	I4	C63D
INIT	C719	IROR	C6B0	LBK30	C6B6	LC120	C61B
LC121	C634	LC122	C641	LC123	C66A	LC124	C66E
LC125	C677	LC126	C680	LC127	C68D	LC128	C691
LC129	C69C	MARST	C6C0	MERET	C7FA	MEVEC	C7F2
MPT26	C6B4	MRDCL	C6F2	MRS10	C6CF	MSGHDR	C7A8
MSGIN	C7D3	PINPLN	C6A0	RESET	C719	RINIT	C7A0
RNDA	C6A8	RNDB	C6AC	SCRST	C6FD	SIPAR	C7E0
SPT02	C6BA	SRS10	C705	STCOL	C7EE	UTRST	C70E

```
002                             ORG   :C7FB
003                     *
004                     *
005                     *
006                     ************************************
007                     * CHECK FOR HIGHEST RAM ADDRESS *
008                     ************************************
009                     *
010                     * Entry: No conditions.
011                     * Exit:  HL points after RAM.
012                     *        BC preserved, ADEF corrupted.
013                     *
014 C7FB 110010         MEMCHK  LXI   D,:1000
015 C7FE 210000                 LXI   H,:0000   Start at 0000
016 C801 19             MCK10   DAD   D         Incr. with #1000
017 C802 7E                     MOV   A,M       Get what is in memory
018 C803 2F                     CMA             Take its complement
019 C804 77                     MOV   M,A       and store it back
020 C805 BE                     CMP   M         Then compare
021 C806 2F                     CMA
022 C807 77                     MOV   M,A       Restore original value
023 C808 CA01C8                 JZ    :C801     Next block if still RAM
024 C80B C9                     RET
025                     *
026                     ************************
027                     * START FROM SCRATCH *
028                     ************************
029                     *
030                     * Entry to Basic monitor.
031                     *
032
033                     * If out of a Hard BREAK:
034
035 C80C 3100F9         RSTART  LXI   SP,:F900  Reset stackpointer
036 C80F CDE4CE                 CALL  :CEE4     Select ROM bank 0,
037 C812 B8DB                   DBL   :DBB8     print '*** BREAK'
038
039                     * Re-enter Basic after run-time error,
040                     * except on input:
041
042 C814 AF             START   XRA   A
043 C815 323501                 STA   :0135     Input from keyb/DINC
044
045                     * Entry on reset, after encoding program line,
046                     * after END:
047
048 C818 2100F9         ST10    LXI   H,:F900
049 C81B 222701                 SHLD  :0127     Reset current base stack
050 C81E F9                     SPHL            Reset stackpointer
051 C81F AF                     XRA   A
052 C820 322601                 STA   :0126     No suspended program
053
054                     * Restart interpreter; entry after end of program,
055                     * after direct command, after soft BREAK, after
056                     * direct command error, after STOP:
057
058 C823 2A2701         ST20    LHLD  :0127     Get saved stackpointer
059 C826 F9                     SPHL            Set it to saved value
060 C827 210000                 LXI   H,:0000
061 C82A 220001                 SHLD  :0100     Reset current line nr.
062 C82D 220401                 SHLD  :0104     No running loops
063 C830 221301                 SHLD  :0113     No active subroutine call
```

```
064 C833 7C                    MOV   A,H
065 C834 322201                STA   :0122     No encoding of stored line
066 C837 00                    NOP
067 C838 00                    NOP
068 C839 00                    NOP
069 C83A 211701                LXI   H,:0117
070 C83D 77                    MOV   M,A       No running of inputs
071 C83E 23                    INX   H
072 C83F 77                    MOV   M,A       No running of program
073 C840 CD88D9                CALL  :D988     Enable keyboard interrupt
074 C843 CDDBD9                CALL  :D9DB     Enable clock interrupt
075 C846 3A3501        ST23    LDA   :0135     Get input direction
076 C849 FE02                  CPI   :02
077 C84B CC79D8                CZ    :D879     EFSW=2: input from editbuf
078 C84E D267C8                JNC   :C867     EFSW>=2: encode ENCODE TEXTLINE IF EDITBUF NOT EMPTY
079 C851 3E2A                  MVI   A,:2A
080 C853 CD1ADD                CALL  :DD1A     Print '*', scan keyboard (ERRATUM DANAMIC 83-17 P231)
081                                            and display characters
082                                            until Break or car.ret
083                                            (If no input is given,
084                                            the DAI remains here in a
085                                            endless loop).
086 C856 DA46C8                JC    :C846     If BREAK: new inputs
087 C859 CDD2DD                CALL  :DDD2     Get char from line, neglect
088                                            TAB and space
089 C85C FE0D                  CPI   :0D
090 C85E CA46C8                JZ    :C846     If car.ret: new inputs
091 C861 CD0DDE                CALL  :DE0D     Check if char is number
092 C864 D26DC8                JNC   :C86D     If no leading nr: encode cmd
093
094                    * Encode program line (if 1st char is number):
095
096 C867 CD18C9         ST24   CALL  :C918     Encode program line, update
097                                            program
098 C86A C318C8                JMP   :C818     Get next input line; kill
099                                            any suspended program
100
101                    * Encode direct command (if 1st char is no number):
102
103 C86D 1680           ST25   MVI   D,:80     Mask for direct command
104 C86F E5                    PUSH  H         Pointer to RUNF
105 C870 213F01                LXI   H,:013F   Addr EBUF
106 C873 E5                    PUSH  H         Save it on stack
107 C874 CF                    RST   1         Encode immediate cmd line
108 C875 00                    DATA  :00
109 C876 3600                  MVI   M,:00     Dummy end of program
110 C878 CD5EDD                CALL  :DD5E     Print car.ret
111 C87B C1                    POP   B         Get EBUF pntr
112 C87C E1                    POP   H         Pntr to RUNF
113 C87D 36FF                  MVI   M,:FF     Set flag running programs
114
115                    * Run a Basic line:
116
117 C87F 0A           ST30     LDAX  B         Get 1st byte from EBUF:
118                                            <  #80: length,
119                                            >= #80: Token.
120 C880 03           ST35     INX   B
121 C881 87                    ADD   A         Calc offset from CF00
122 C882 D2E5C8                JNC   :C8E5     Jump if length byte
123 C885 6F                    MOV   L,A       Get table address in HL
124 C886 26CF                  MVI   H,:CF
125 C888 7E                    MOV   A,M       ) Get addr Basic routine
```

```
126 C889 23                 INX   H          ) from table in HL
127 C88A 66                 MOV   H,M        )
128 C88B 6F                 MOV   L,A        )
129 C88C CDA9C8             CALL  :C8A9      Perform this routine
130
131                 * Commands return here:
132
133 C88F DAAAC8     ENDCOM  JC    :C8AA      Jump if special action
134
135                 * If suspended:
136
137 C892 60                 MOV   H,B
138 C893 69                 MOV   L,C
139 C894 220201             SHLD  :0102      Remember start next cmd
140 C897 3AC402             LDA   :02C4
141 C89A B7                 ORA   A          BREAK flag set?
142 C89B CA7FC8             JZ    :C87F      Run Basic line if not
143 C89E 00                 NOP
144 C89F 00                 NOP
145 C8A0 00                 NOP
146 C8A1 3EFF               MVI   A,:FF
147 C8A3 32C402             STA   :02C4      Set BREAK flag 'serviced'
148 C8A6 C3C0C8             JMP   :C8C0      Handle break
149
150                 * Run a BASIC line:
151
152 C8A9 E9         DCALL   PCHL             Addr Basic routine in PC
153
154                 * If special end of action:
155
156 C8AA FE02         ST40    CPI   :02
157 C8AC CAC0C8             JZ    :C8C0      If soft break (2)
158 C8AF D2B8C8             JNC   :C8B8      If STOP (3)
159 C8B2 EA18C8             JPE   :C818      If can't continu (1)
160 C8B5 C308C9             JMP   :C908      If after LOAD (0)
161
162                 * If 'STOP':
163
164 C8B8 60         ST45    MOV   H,B
165 C8B9 69                 MOV   L,C
166 C8BA 220201             SHLD  :0102      Remember where next cmd
167 C8BD C3C5C8             JMP   :C8C5      Print 'IN LINE ..' and
168                                          handle a break
169
170                 * If suspended (soft Break handling):
171
172 C8C0 CDFFDA     ST50    CALL  :DAFF      Print car.ret; 'BREAK'.
173 C8C3 C5DB               DBL   :DBC5
174 C8C5 CD75DA     ST55    CALL  :DA75      Print 'IN LINE .....'
175                                          or car.ret
176 C8C8 CA23C8             JZ    :C823      Jump if immediate cmd
177
178                 * Only if 'break' in program:
179
180 C8CB 21EBFF             LXI   H,:FFEB    Frame length
181 C8CE 39                 DAD   SP         New stack level
182 C8CF 44                 MOV   B,H
183 C8D0 4D                 MOV   C,L
184 C8D1 222701             SHLD  :0127      Set new base stack
185 C8D4 F9                 SPHL             Set stackpointer
186 C8D5 110001             LXI   D,:0100    ) Boundaries frame
187 C8D8 211501             LXI   H,:0115    )
```

```
188 C8DB CD4FDE             CALL  :DE4F     Save program status
189                                         (FRAME) on stack.
190 C8DE 212601             LXI   H,:0126
191 C8E1 34                 INR   M         Set flag existence saved
192                                         program
193 C8E2 C323C8             JMP   :C823     Run again
194
195                 * Length byte or end flag:
196
197 C8E5 CA23C8      ST60   JZ    :C823     If end immediate cmd
198                                         line or end program
199 C8E8 60                 MOV   H,B
200 C8E9 69                 MOV   L,C
201 C8EA 220001             SHLD  :0100     Store start current line
202 C8ED 2A1501             LHLD  :0115     Get trace + step flag
203 C8F0 7C                 MOV   A,H
204 C8F1 B5                 ORA   L
205 C8F2 CA00C9             JZ    :C900     If no step/trace flag
206
207                 * If step/trace flag set:
208
209 C8F5 E5                 PUSH  H
210 C8F6 C5                 PUSH  B
211 C8F7 CDA4CE             CALL  :CEA4     List current line
212 C8FA C1                 POP   B
213 C8FB F1                 POP   PSW       Get step flag
214 C8FC B7                 ORA   A         If set:
215 C8FD C4DAD6             CNZ   :D6DA     Wait for spacebar pressed
216 C900 03          ST65   INX   B
217 C901 03                 INX   B         Pnts after linenr
218 C902 DACBC8             JC    :C8CB     If Break
219 C905 C37FC8             JMP   :C87F     Run next BASIC line
220
221                 * Special action after LOAD:
222
223 C908 2A0001      ST70   LHLD  :0100     Get start current line
224 C90B 7C                 MOV   A,H
225 C90C B5                 ORA   L         Direct cmd?
226 C90D CA7FC8             JZ    :C87F     Then run Basic line
227 C910 3100F9             LXI   SP,:F900  Else: reset stackpointer
228 C913 3E87               MVI   A,:87     Simulate Token 'RUN'
229 C915 C380C8             JMP   :C880     Pretend RUN cmd
230                 *
231                 * PROGRAM INPUT:
232                 *
233                 * Encodes a program line and updates the stored
234                 * program.
235                 *
236                 * Entry: C: Input count / offset.
237                 * Exit:  C: Offset after line.
238                 *        AFBDEHL preserved.
239                 *
240 C918 213F01      PROGI  LXI   H,:013F   Addr buf for encoded cmds
241 C91B CF                 RST   1         Get linenr
242 C91C 03                 DATA  :03
243 C91D CDD2DD             CALL  :DDD2     Get char from line;
244                                         neglect TAB + space
245 C920 FE0D               CPI   :0D       Car.ret ?
246 C922 CCA2C9             CZ    :C9A2     Delete old version if
247                                         only linenr given
248 C925 CA3BC9             JZ    :C93B     Jump if linenr only
249 C928 D5                 PUSH  D         Remember linenr
```

```
C929 1640          MVI   D,:40     Mask for 'stored cmd'
C92B CD3CC9        CALL  :C93C     Encode a line
C92E 7D            MOV   A,L
C92F D63F          SUI   :3F       Length string in A
C931 323E01        STA   :013E     Length in EBUF
C934 D1            POP   D         Get linenr
C935 CDA2C9        CALL  :C9A2     Delete old line
C938 CDBDC9        CALL  :C9BD     Insert new line
C93B C9      PGI20 RET
                   *
                   * ENCODE A LINE:
                   *
                   * Exit: DE restored.
                   *       HL points to 1st free byte in EBUF.
                   *        C points after car.ret in input.
                   *        A=0, F corrupted.
                   *
C93C D5      ELINA PUSH  D
C93D C5            PUSH  B
C93E E5            PUSH  H
C93F 210000        LXI   H,:0000
C942 39            DAD   SP        Stackpointer in HL
C943 221D01        SHLD  :011D     Save stackpointer
C946 E1            POP   H
C947 E5            PUSH  H
C948 3E01          MVI   A,:01
C94A 322201        STA   :0122     Set encoding a stored line
C94D CF            RST   1         Encode inputs
C94E 00            DATA  :00
C94F D1            POP   D         ) Cancel Push B,H
C950 D1            POP   D         )
C951 AF      ELA10 XRA   A
C952 322201        STA   :0122     No encoding stored line
C955 D1            POP   D
C956 C9            RET
                   *
                   ****************************************
                   * ERROR WHILE ENCODING A STORED LINE *
                   ****************************************
                   *
                   * Restores stackpointer, adds '***' to begin of
                   * line, adds '?' to place of error. Line is entered
                   * into the encoded inputbuffer (EBUF).
                   *
                   * Entry: B: Errorcode.
                   *        C: Place of error.
                   *        On stack: BC points to input.
                   *                  HL points to EBUF.
                   *
C957 2A1D01  ELARS LHLD  :011D     Get ERSSP
C95A F9            SPHL            Restore stackpointer
C95B E1            POP   H         Get buffer pointer
C95C 78            MOV   A,B       Errorcode in A
C95D 51            MOV   D,C       Place of error in D
C95E C1            POP   B         Get input pointer
C95F 47            MOV   B,A
C960 36B1          MVI   M,:B1     Token for '***' in EBUF
C962 23            INX   H
C963 E5            PUSH  H         Save buf pointer
C964 23            INX   H
C965 79      ELA20 MOV   A,C       ) Place of error
C966 BA            CMP   D         ) reached ?
```

```
312 C967 3E3F              MVI   A,:3F
313 C969 CC95C9            CZ    :C995      Then insert '?'
314 C96C CDE0DD            CALL  :DDE0      Get char. from line
315 C96F 0C                INR   C          Update inputpointer
316 C970 FE0D              CPI   :0D        Line done ?
317 C972 C495C9            CNZ   :C995      Insert char in EBUF if not
318 C975 C265C9            JNZ   :C965      Next char if not ready
319 C978 7D                MOV   A,L        Lobyte EBUF pntr in A
320 C979 D1                POP   D          Addr after '***'
321 C97A 93                SUB   E
322 C97B 3D                DCR   A
323 C97C 12                STAX  D          Store length in EBUF
324 C97D 3600              MVI   M,:00      0 after string
325 C97F C5                PUSH  B          Save errormessage pntr
326 C980 42                MOV   B,D        ) EBUF pntr in BC
327 C981 4B                MOV   C,E        )
328 C982 0B                DCX   B
329 C983 0B                DCX   B
330 C984 0B                DCX   B
331 C985 0B                DCX   B          Pnts to begin EBUF
332 C986 CD5EDD            CALL  :DD5E      Print car.ret
333 C989 CDABEC            CALL  :ECAB      (0) List current line
334 C98C C1                POP   B          Get errormessage pntr
335 C98D E5                PUSH  H
336 C98E CD50DA            CALL  :DA50      Print errormessage
337 C991 E1                POP   H
338 C992 C351C9            JMP   :C951      Store 0 in ERSFL, Pop D,
339                                         and ret.
340                  *
341                  * INSERT CHARACTER IN ENCODED INPUT BUFFER:
342                  *
343                  * A character is inserted in the EBUF only if
344                  * there is space available.
345                  *
346                  * Entry: HL: 1st free location in EBUF.
347                  *        A:  Character to be inserted.
348                  * Exit:  HL updated. AFBCDE preserved.
349                  *
350 C995 F5           ELAIN  PUSH  PSW
351 C996 7D                   MOV   A,L        Get lobyte of EBUF pntr
352 C997 FEBC                CPI   :BC        Buffer full ?
353 C999 CAA0C9              JZ    :C9A0      Then abort
354 C99C F1                 POP   PSW
355 C99D 77                 MOV   M,A        Char into EBUF
356 C99E 23                 INX   H          Update pntr
357 C99F C9                 RET
358
359                  * If EBUF full:
360
361 C9A0 F1           EAI10  POP   PSW        No action
362 C9A1 C9                 RET
363                  *
364                  **********************************
365                  * DELETE OLD VERSION OF A LINE *
366                  **********************************
367                  *
368                  * A textline is deleted by moving the rest of the
369                  * textbuffer and the symboltable 'downwards'.
370                  *
371                  * Entry: DE: requested linenumber.
372                  * Exit:  DE points to linenr after deleted line.
373                  *        AFBCHL preserved.
```

```
374                     *
375 C9A2 F5             LDEL    PUSH  PSW
376 C9A3 C5                     PUSH  B
377 C9A4 E5                     PUSH  H
378 C9A5 EB                     XCHG            Linenr in HL
379 C9A6 CDF6CA                 CALL  :CAF6     Addr line in textbuf in HL
380 C9A9 D2B8C9                 JNC   :C9B8     Abort if not found
381 C9AC 7E                     MOV   A,M       Get line length
382 C9AD 2F                     CMA
383 C9AE 5F                     MOV   E,A       Compl. value in E
384 C9AF 16FF                   MVI   D,:FF
385 C9B1 CD39DE                 CALL  :DE39     HL=HL-line length
386 C9B4 EB                     XCHG
387 C9B5 CDD1C9                 CALL  :C9D1     Move program buffers
388 C9B8 EB             LDL10   XCHG
389 C9B9 E1                     POP   H
390 C9BA C1                     POP   B
391 C9BB F1                     POP   PSW
392 C9BC C9                     RET
393                     *
394                     **********************
395                     * INSERT A NEW LINE *
396                     **********************
397                     *
398                     * Inserts a encoded line in the textbuffer.
399                     * Required space for the textline is made by
400                     * shifting the rest of the textbuffer and the
401                     * symboltable 'upwards'.
402                     *
403                     * Entry: DE: Destination address in textbuffer.
404                     *        HL: Points after string in EBUF.
405                     *        A:  Length string in EBUF.
406                     *
407 C9BD C5             LINS    PUSH  B
408 C9BE E5                     PUSH  H
409 C9BF 6F                     MOV   L,A
410 C9C0 2600                   MVI   H,:00     String length in HL
411 C9C2 23                     INX   H         Required space in HEAP
412 C9C3 D5                     PUSH  D
413 C9C4 CDD1C9                 CALL  :C9D1     Move program buffers
414 C9C7 C1                     POP   B
415 C9C8 E1                     POP   H
416 C9C9 113E01                 LXI   D,:013E   Startaddr. EBUF
417 C9CC CD4FDE                 CALL  :DE4F     Transfer data from EBUF
418                                             into textbuffer
419 C9CF C1                     POP   B
420 C9D0 C9                     RET
421                     *
422                     ************************
423                     * MOVE PROGRAM BUFFERS *
424                     ************************
425                     *
426                     * Moves a part (or the whole) textbuffer and the
427                     * whole symboltable up or down.
428                     * The startaddress of the textbuffer and the end
429                     * of the symboltable are set depending on the
430                     * heap size. The heap pointers are updated.
431                     *
432                     * Entry: DE: Address from where to update.
433                     *        HL: Length of area to be inserted/deleted.
434                     * Entry if running 'NEW' only:
435                     *        BC: 0.
```

```
436                     *         DE: startaddress Heap.
437                     *         HL: Size Heap.
438                     *         At entry, the pointers for textbuf and
439                     *         symtab are as if HEAPsize is zero.
440                     * Exit:   AFBCDE preserved.
441                     *         HL: New startaddress.
442                     *
443 C9D1 C5            PROGM   PUSH  B
444 C9D2 D5                    PUSH  D
445 C9D3 42                    MOV   B,D       ) Addr from where to
446 C9D4 4B                    MOV   C,E       ) update in BC
447 C9D5 EB                    XCHG            Length area in DE
448 C9D6 2AA302                LHLD  :02A3     Get end symtab
449 C9D9 E5                    PUSH  H
450 C9DA D5                    PUSH  D
451 C9DB 19                    DAD   D         New end symtab
452 C9DC EB                    XCHG
453 C9DD 2AA502                LHLD  :02A5     Get bottom screen RAM
454 C9E0 CD14DE                CALL  :DE14     Check for overflow
455 C9E3 EB                    XCHG            End symtab in HL
456 C9E4 DA10DA                JC    :DA10     Evt. run 'OUT OF MEMORY'
457 C9E7 22A302                SHLD  :02A3     Store end symbol table
458 C9EA D1                    POP   D
459 C9EB D5                    PUSH  D
460 C9EC 2AA102                LHLD  :02A1     Get begin symtab
461 C9EF 19                    DAD   D         New begin symtab
462 C9F0 22A102                SHLD  :02A1     Store begin symbol table
463 C9F3 E1            PRGM1   POP   H
464 C9F4 50                    MOV   D,B
465 C9F5 59                    MOV   E,C       Startaddr in DE
466 C9F6 19                    DAD   D         New startaddr
467 C9F7 44                    MOV   B,H
468 C9F8 4D                    MOV   C,L       New startaddr in BC
469 C9F9 E3                    XTHL            Get old end symtab
470 C9FA CD4FDE                CALL  :DE4F     Move textbuf + symtab
471                                            from old to new addr.
472 C9FD E1                    POP   H
473 C9FE D1                    POP   D
474 C9FF C1                    POP   B
475 CA00 C9                    RET
476                     *
477                     *
478                     *
479 CA01                       END
```

```
****************************
* S Y M B O L   T A B L E *
****************************

DCALL   C8A9    EAI10   C9A0    ELA10   C951    ELA20   C965
ELAIN   C995    ELARS   C957    ELINA   C93C    ENDCOM  C88F
LDEL    C9A2    LDL10   C9B8    LINS    C9BD    MCK10   C801
MEMCHK  C7FB    PGI20   C93B    PRGM1   C9F3    PROGI   C918
PROGM   C9D1    RSTART  C80C    ST10    C818    ST20    C823
ST23    C846    ST24    C867    ST25    C86D    ST30    C87F
ST35    C880    ST40    C8AA    ST45    C8B8    ST50    C8C0
ST55    C8C5    ST60    C8E5    ST65    C900    ST70    C908
START   C814
```

```
002                             ORG   :CA01
003                     *
004                     *
005                     *
006                     ******************************
007                     * MEMORY MANAGEMENT ROUTINE *
008                     ******************************
009                     *
010                     * This routine is used to obtain and release
011                     * memory for its display, as the mode changes.
012                     *
013                     * Entry: HL: Lowest screen RAM byte required.
014                     *        CY=1: Space to this point at least is
015                     *              now required. Additional space
016                     *              may not be released.
017                     *        CY=0: Space is held to or below this
018                     *              point. Any space held below this
019                     *              point is no longer required.
020                     * Exit:  CY=1: O.K.
021                     *        CY=0: No space available.
022                     *        AFBCDE preserved.
023                     *
024                     ASKRM
025 CA01 00             SMKRM   NOP
026 CA02 F5                     PUSH  PSW
027 CA03 D5                     PUSH  D
028 CA04 D21BCA                 JNC   :CA1B     If space not reqd anymore
029 CA07 EB                     XCHG            Lowest byte in DE
030 CA08 2AA502                 LHLD  :02A5     Get bottom screen RAM
031 CA0B CD14DE                 CALL  :DE14     Check for overflow
032 CA0E DA1ECA                 JC    :CA1E     If OK
033 CA11 2AA302                 LHLD  :02A3     Get begin free RAM
034 CA14 EB                     XCHG            and store it in DE
035 CA15 CD14DE                 CALL  :DE14     Still free RAM available?
036 CA18 DA21CA                 JC    :CA21     If not
037 CA1B 22A502         ASR10   SHLD  :02A5     Update bottom screen RAM
038 CA1E C3ADCE         ASR20   JMP   :CEAD     Return, set CY=1
039
040                     * If no RAM space available:
041
042 CA21 D1             ASR30   POP   D
043 CA22 C3B1CE                 JMP   :CEB1     Return, set CY=0
044                     *
045                     ********************************************
046                     * EMERGENCY STOP ROUTINE (Graphics modes) *
047                     ********************************************
048                     *
049                     * This routine is used if no sufficient space for
050                     * a A-mode is available.
051                     *
052 CA25 CDC6CE         EMSTP   CALL  :CEC6     Set up HEAPsize + buffers
053                                             to default values
054 CA28 3EFF                   MVI   A,:FF
055 CA2A EF                     RST   5         Change to mode 0
056 CA2B 18                     DATA  :18
057 CA2C CDE4CE                 CALL  :CEE4     Select ROM-bank 0; Run error
058 CA2F 89DB                   DBL   :DB89     'OUT OF SPACE FOR MODE'
059 CA31 C318C8                 JMP   :C818     Return to BASIC monitor
060                     *
061                     ******************************************
062                     * FIND STRING BASIC INSTRUCTION IN TABLE *
063                     ******************************************
```

```
064                     *
065                     * Looks for table entry whose name is the initial
066                     * string of input, beginning at C'th position.
067                     * REMARK: Variables, beginning with a reserved
068                     *         string are not allowed.
069                     *
070                     * Entry: HL: Startaddress table.
071                     *        C : Position char on current line.
072                     *        E : Number of info bytes -1.
073                     * Exit:   If found: CY=1:
074                     *             HL: Address in table where string
075                     *                 can be found.
076                     *             C : Position on current line after
077                     *                 last char.
078                     *             A : Last byte of string typed in.
079                     *             BE preserved, D=0.
080                     *         If not found: CY=0:
081                     *             C : Points to next char.
082                     *             HL: Points after end of table.
083                     *             BE preserved. A,D=0.
084                     *
085 CA34 CDD2DD      LOOKC   CALL  :DDD2     Get char from line,
086                                         neglect TAB and space
087 CA37 56          LKC10   MOV   D,M       Get length byte of string
088 CA38 23                  INX   H         Points to 1st stringchar.
089 CA39 7A                  MOV   A,D
090 CA3A B7                  ORA   A         Is length zero?
091 CA3B C8                  RZ              Then abort
092 CA3C C5                  PUSH  B         Save position of 1st char
093 CA3D CDE0DD      LKC20   CALL  :DDE0     Get char from line
094 CA40 0C                  INR   C         Points to next char on line
095 CA41 BE                  CMP   M         Is it identical to the
096                                         one in table?
097 CA42 23                  INX   H         Points to next char in table
098 CA43 C24ECA              JNZ   :CA4E     If not identical
099 CA46 15                  DCR   D         Else: decr string length
100 CA47 C23DCA              JNZ   :CA3D     Get evt. next byte to check
101 CA4A E3                  XTHL            cancell PUSH B
102 CA4B E1                  POP   H
103 CA4C 37                  STC             CY=1
104 CA4D C9                  RET
105
106                     * If strings not identical:
107
108 CA4E 7A          LKC30   MOV   A,D       Get string length
109 CA4F 83                  ADD   E         Add 2
110 CA50 CD30DE              CALL  :DE30     Add A to HL; HL points now
111                                         to next string in table.
112 CA53 C1                  POP   B         Restore C=1
113 CA54 C337CA              JMP   :CA37     Start check on next string
114                     *
115                     ******************
116                     * TABLE LOOK UP *
117                     ******************
118                     *
119                     * Finds an entry in a look-up table.
120                     * LOOK used for symboltable, LOOKX for table of
121                     * Basic functions (CFE6).
122                     *
123                     * Table format:
124                     * [type/length][name][type/length][info]
125                     *
```

(ERRATUM DYNAMIC 83-17 P 231)

```
                        * Entry: BC: Points to start name in input.
                        *        D:  Type/length of name in input
                        *            high nibble: type; low nibble: length.
                        *        HL: Startaddress look-up table.
                        * Exit:   If not found: CY=0:
                        *            HL points to 0 byte at table end.
                        *            ABCD preserved, E corrupted.
                        *        If found: CY=1:
                        *            HL points to T/L of entry found.
                        *            E indicates how manyth entry.
                        *            ABC preserved.
                        *
CA57 2AA102     LOOK    LHLD  :02A1     Get startaddr symtab
                *
CA5A 37         LOOKX   STC              CY=1
CA5B F5                 PUSH  PSW
CA5C C5                 PUSH  B
CA5D 1EFF               MVI   E,:FF
CA5F 1C         LK10    INR   E          Entry count
CA60 7E                 MOV   A,M        Get T/L name in table
CA61 B7                 ORA   A
CA62 CA8BCA             JZ    :CA8B      Abort if end table reached
CA65 BA                 CMP   D          Compare with wanted T/L
CA66 CA6FCA             JZ    :CA6F      Jump if found
CA69 CDAECA     LK15    CALL  :CAAE      Calc addr next entry
CA6C C35FCA             JMP   :CA5F      Check next entry

                * If T/L of name O.K.:

CA6F C1         LK20    POP   B
CA70 C5                 PUSH  B
CA71 48                 MOV   C,B
CA72 D5                 PUSH  D
CA73 7A                 MOV   A,D        Get wanted T/L of name
CA74 E60F               ANI   :0F        Length name only
CA76 57                 MOV   D,A        in D
CA77 E5                 PUSH  H          Save begin table entry
CA78 23         LK30    INX   H
CA79 CDE0DD             CALL  :DDE0      Get char from line
CA7C BE                 CMP   M          Compare char of name
CA7D C28FCA             JNZ   :CA8F      If not correct name

                * If char. identical:

CA80 0C                 INR   C          Points to next char
CA81 15                 DCR   D          Decr length
CA82 C278CA             JNZ   :CA78      Check next char if not
                                         ready
CA85 23                 INX   H          Points after name in table
CA86 D1                 POP   D
CA87 D1                 POP   D
CA88 C1                 POP   B
CA89 F1                 POP   PSW        CY=1: Entry found
CA8A C9                 RET

                * If end of look-up table reached:

CA8B C1         LK40    POP   B
CA8C F1                 POP   PSW
CA8D 3F                 CMC              CY=0: No entry found
CA8E C9                 RET
```

```
188                     * If characters not identical:
189                     *
190 CA8F E1             LK50    POP   H
191 CA90 D1                     POP   D
192 CA91 7A                     MOV   A,D       Length in A
193 CA92 C369CA                 JMP   :CA69     Look further
194                     *
195                     ****************************************
196                     * FIND A VARIABLE IN THE SYMBOLTABLE *
197                     ****************************************
198                     *
199                     * Routine skips through successive symtab entries
200                     * from beginning till past the place pointed by HL.
201                     *
202                     * Entry: HL points to 1st byte required variable.
203                     * Exit:  HL points to (if found) or past (if not
204                     *        found) address required in symbol table.
205                     *        AFBCDE preserved.
206                     *
207 CA95 F5             FNAME   PUSH  PSW
208 CA96 D5                     PUSH  D
209 CA97 EB                     XCHG            Reqd addr in DE
210 CA98 2AA102                 LHLD  :02A1     Get startaddr symtab
211 CA9B E5             FNM10   PUSH  H
212 CA9C CDAECA                 CALL  :CAAE     Calc addr next variable
213 CA9F CD14DE                 CALL  :DE14     Reqd variable reached ?
214 CAA2 D2AACA                 JNC   :CAAA     Quit if true
215 CAA5 33                     INX   SP        ) Cancel Push H
216 CAA6 33                     INX   SP        )
217 CAA7 C39BCA                 JMP   :CA9B     Skip next variable
218 CAAA E1             FNM20   POP   H
219 CAAB D1                     POP   D
220 CAAC F1                     POP   PSW
221 CAAD C9                     RET
222                     *
223                     ******************************************************
224                     * CALCULATE ADDRESS NEXT VARIABLE IN SYMBOLTABLE *
225                     ******************************************************
226                     *
227                     * Adds length of name of variable + length of value
228                     * of variable to beginaddress.
229                     *
230                     * DADD: variable = length/name/length/value.
231                     * DADR: variable = length/name.
232                     *
233                     * Entry: HL points to 1st byte of current variable.
234                     * Exit:  HL points to next variable in symtab.
235                     *
236 CAAE CDB1CA    DADD    CALL  :CAB1     Add length name to HL
237 CAB1 7E        DADR    MOV   A,M       Get info T/L byte
238 CAB2 23                INX   H         Add 1
239 CAB3 E60F              ANI   :0F       Length only
240 CAB5 C330DE            JMP   :DE30     Add length info to HL
241                     *
242                     ************************************************
243                     * INSERT A VARIABLE NAME IN THE SYMBOL TABLE *
244                     ************************************************
245                     *
246                     * Entry: HL points to end symtab.
247                     *        B  points to start of name in input.
248                     *        E  number of bytes of info to reserve.
249                     *        D  T/L byte of name.
```

```
250                     * Exit:  HL points to info T/L byte.
251                     *        AFBCDE preserved.
252                     *
253 CAB8 F5             LOOKI   PUSH  PSW
254 CAB9 C5                     PUSH  B
255 CABA 48                     MOV   C,B       Input pos. in C
256 CABB D5                     PUSH  D
257 CABC D5                     PUSH  D
258 CABD E5                     PUSH  H
259 CABE 7A                     MOV   A,D       T/L name in A
260 CABF E60F                   ANI   :0F       Name length only
261 CAC1 83                     ADD   E         Add length info
262 CAC2 3C                     INR   A
263 CAC3 3C                     INR   A         +2 (length new entry)
264 CAC4 CD30DE                 CALL  :DE30     Calc new end symtab -1
265 CAC7 EB                     XCHG
266 CAC8 2AA502                 LHLD  :02A5     Get bottom screen RAM
267 CACB EB                     XCHG
268 CACC CD14DE                 CALL  :DE14     Compare DE-HL
269 CACF 3E1B                   MVI   A,:1B
270 CAD1 D2F5D9                 JNC   :D9F5     Run 'OUT OF MEMORY' if
271                                             not sufficient free RAM
272 CAD4 3600                   MVI   M,:00     New 'end table' flag
273 CAD6 23                     INX   H         HL is new end symtab
274 CAD7 22A302                 SHLD  :02A3     Store end symtab
275 CADA E1                     POP   H         Get old end symtab
276 CADB D1                     POP   D         Get T/L info
277 CADC 72                     MOV   M,D       Into symtab
278 CADD 23                     INX   H
279 CADE 7A                     MOV   A,D
280 CADF E670                   ANI   :70       Get type only
281 CAE1 B3                     ORA   E         Set low nibble for length
282 CAE2 5F                     MOV   E,A
283 CAE3 7A                     MOV   A,D       Get length name
284 CAE4 E60F                   ANI   :0F       Max. 15 bytes
285 CAE6 57                     MOV   D,A       Length in D for count
286 CAE7 CDE0DD         LKI10   CALL  :DDE0     Get char from line
287 CAEA 77                     MOV   M,A       Char into symtab
288 CAEB 0C                     INR   C         Pnts to next char on line
289 CAEC 23                     INX   H         Next pos in symtab
290 CAED 15                     DCR   D         Decr length name
291 CAEE C2E7CA                 JNZ   :CAE7     Next char if not ready
292 CAF1 73                     MOV   M,E       Info T/L into symtab
293 CAF2 D1                     POP   D
294 CAF3 C1                     POP   B
295 CAF4 F1                     POP   PSW
296 CAF5 C9                     RET
297                     *
298                     ************************************
299                     * FIND LINENUMBER IN TEXTBUFFER *
300                     ************************************
301                     *
302                     * Entry: HL: requested linenumber.
303                     * Exit:  ABCDE preserved. F corrupted.
304                     *        CY=1: Linenr found:
305                     *              HL points to address textline.
306                     *        CY=0: Not found:
307                     *              Z=0: HL points to address textline
308                     *                   with next higher linenumber.
309                     *              Z=1: End of textbuffer reached.
310                     *
311 CAF6 C5             FINDL   PUSH  B
```

```
312 CAF7 F5              PUSH  PSW
313 CAF8 D5              PUSH  D
314 CAF9 0600            MVI   B,:00
315 CAFB EB              XCHG              Req. linenr in DE
316 CAFC 2A9F02          LHLD  :029F       Get startaddr textbuf
317 CAFF 48        FDL05 MOV   C,B         Length prev. instr in C
318 CB00 0600            MVI   B,:00
319 CB02 09              DAD   B           Add this length to beginaddr
320 CB03 46        FDL10 MOV   B,M         Get length current instr
321 CB04 78              MOV   A,B         in A
322 CB05 B7              ORA   A
323 CB06 23              INX   H           Pnts to hibyte linenr
324 CB07 CA1DCB          JZ    :CB1D       Abort if at end textbuf
325 CB0A 7A              MOV   A,D         Get hibyte reqd linenr.
326 CB0B BE              CMP   M           Test high order bits
327 CB0C DA1CCB          JC    :CB1C       Abort if reqd nr lower than
328                                        current one (nr > reqd)
329 CB0F C2FFCA          JNZ   :CAFF       Next textline (nr < reqd)
330 CB12 23              INX   H           Pnts to lobyte linenr
331 CB13 7B              MOV   A,E         Get lobyte reqd linenr
332 CB14 BE              CMP   M
333 CB15 2B              DCX   H
334 CB16 DA1CCB          JC    :CB1C       Abort if reqd nr lower than
335                                        current one (nr > reqd)
336 CB19 C2FFCA          JNZ   :CAFF       Next textline if not found
337                                        (nr < reqd)
338 CB1C 3F        FDL20 CMC               Line found: CY=1
339 CB1D 2B        FDL30 DCX   H
340 CB1E D1              POP   D
341 CB1F C1              POP   B
342 CB20 78              MOV   A,B
343 CB21 C1              POP   B
344 CB22 C9              RET
345                 *
346                 ********************************
347                 * EMPTY SYMBOLTABLE AND HEAP *
348                 ********************************
349                 *
350                 * Zeroes all variables, all pointers in the symtab,
351                 * kill all arrays, strings or stringarrays (zero the
352                 * pointers) referenced by the symtab by setting the
353                 * msb of the sizebit =1. Basic program is moved to a
354                 * location corresponding to the Heapsize.
355                 *
356                 * Exit: All registers preserved.
357                 *
358 CB23 F5         SCRATC PUSH  PSW
359 CB24 C5                PUSH  B
360 CB25 D5                PUSH  D
361 CB26 E5                PUSH  H
362 CB27 2AA102           LHLD  :02A1       Get startaddr symtab
363 CB2A 7E        SCT10  MOV   A,M         ) get name length
364 CB2B E60F             ANI   :0F         )
365 CB2D CA53CB           JZ    :CB53       Abort if at end symtab
366 CB30 CDB1CA           CALL  :CAB1       HL pnts to info length byte
367 CB33 7E               MOV   A,M         ) Get type
368 CB34 E6F0             ANI   :F0         )
369 CB36 FE40             CPI   :40
370 CB38 D24DCB           JNC   :CB4D       If array
371 CB3B 23               INX   H
372 CB3C FE20             CPI   :20
373 CB3E CA47CB           JZ    :CB47       If string
```

```
374
375                     * If numeric variable:
376
377 CB41 CD9ECB               CALL  :CB9E     Set value is 0 (4 bytes)
378 CB44 C32ACB               JMP   :CB2A     Next entry
379
380                     * If string variable:
381
382 CB47 CDA8CB       SCT30   CALL  :CBA8     Erase string reference in
383                                             symbtab and Heap
384 CB4A C32ACB               JMP   :CB2A     Next entry
385
386                     * If array:
387
388 CB4D CD5BCB       SCT40   CALL  :CB5B     Erase array
389 CB50 C32ACB               JMP   :CB2A     Next entry
390
391                     * If ready:
392
393 CB53 CDCADE       SCT90   CALL  :DECA     Organise HEAP + buffers
394 CB56 E1           LC188   POP   H
395 CB57 D1                   POP   D
396 CB58 C1                   POP   B
397 CB59 F1                   POP   PSW
398 CB5A C9                   RET
399                     *
400                     ***************
401                     * ERASE ARRAY *
402                     ***************
403                     *
404                     * The pointer is zeroed, the array size is erased
405                     * (msb=1). For stringarrays: zeroes all pointers
406                     * in the array and erase the string in the Heap.
407                     *
408                     * Entry: HL points to T/L byte of info after a
409                     *           symtab name of an array.
410                     * Exit:  HL points to next symtab entry.
411                     *        AFBCDE preserved.
412                     *
413 CB5B D5           EARRAY  PUSH  D
414 CB5C C5                   PUSH  B
415 CB5D F5                   PUSH  PSW
416 CB5E 7E                   MOV   A,M       Get type info
417 CB5F E630                 ANI   :30
418 CB61 F5                   PUSH  PSW       Save type only
419 CB62 CD51CE               CALL  :CE51     ) Get addr of array in
420 CB65 23                   INX   H         ) Heap in DE,
421 CB66 56                   MOV   D,M       ) Kill pointer in the
422 CB67 3600                 MVI   M,:00     ) symboltable
423 CB69 23                   INX   H         Pnts after symtab entry
424 CB6A 7A                   MOV   A,D
425 CB6B B3                   ORA   E
426 CB6C CA99CB               JZ    :CB99     Abort if entry was already 0
427 CB6F EB                   XCHG            Arrayaddr in HL
428 CB70 2B                   DCX   H
429 CB71 2B                   DCX   H         Pnts to 1st byte Heap entry
430 CB72 46                   MOV   B,M       Get 1st byte
431 CB73 CD36D2               CALL  :D236     Clear heap entry by msb=1
432 CB76 F1                   POP   PSW       Get type info
433 CB77 F5                   PUSH  PSW
434 CB78 FE20                 CPI   :20       String ?
435 CB7A D5                   PUSH  D         Save stringpointer
```

```
436 CB7B C298CB              JNZ   :CB98     Abort if not string
437
438                  * If string array:
439
440 CB7E 23                   INX   H
441 CB7F 4E                   MOV   C,M       Get length Heap entry
442 CB80 23                   INX   H
443 CB81 5E                   MOV   E,M       Get dimension
444 CB82 1C                   INR   E
445 CB83 7B                   MOV   A,E
446 CB84 CD30DE               CALL  :DE30     Calc beginaddr stringpntrs
447 CB87 79                   MOV   A,C
448 CB88 93                   SUB   E         Calc length pntr area
449 CB89 4F                   MOV   C,A       in C
450 CB8A D28ECB               JNC   :CB8E
451 CB8D 05                   DCR   B
452 CB8E CDA8CB      EAR10    CALL  :CBA8     Erase stringreference in
453                                            symtab and Heap
454 CB91 0B                   DCX   B
455 CB92 0B                   DCX   B         Update length pntr area
456 CB93 78                   MOV   A,B
457 CB94 B1                   ORA   C
458 CB95 C28ECB               JNZ   :CB8E     Next string if not ready
459
460                  * If ready:
461
462 CB98 E1           EAR20   POP   H         Get symtab pntr
463 CB99 F1           EAR30   POP   PSW
464 CB9A F1                   POP   PSW
465 CB9B C1                   POP   B
466 CB9C D1                   POP   D
467 CB9D C9                   RET
468                  *
469                  ************************************************
470                  * CLEAR A NUMERIC VARIABLE IN THE SYMBOLTABLE *
471                  ************************************************
472                  *
473                  * Loads '0' into 4 consecutive memory locations.
474                  *
475                  * Entry: Startaddress in HL.
476                  * Exit:  A=0, HL points to next byte.
477                  *        BCDEF preserved.
478                  *
479 CB9E AF           ZFPINT  XRA   A
480 CB9F 77                   MOV   M,A
481 CBA0 23                   INX   H
482 CBA1 77                   MOV   M,A
483 CBA2 23                   INX   H
484 CBA3 77                   MOV   M,A
485 CBA4 23                   INX   H
486 CBA5 77                   MOV   M,A
487 CBA6 23                   INX   H
488 CBA7 C9                   RET
489                  *
490                  ********************************************
491                  * ERASE STRINGREFERENCE IN HEAP AND SYMTAB *
492                  ********************************************
493                  *
494                  * The pointer in the symtab is set to '0', the
495                  * msb of the sizebyte of the Heap entry is set
496                  * to 1.
497                  *
```

```
498                     * Entry: HL points to stringpointer in symtab.
499                     * Exit:  HL points after this pointer.
500                     *        DE stringpointer.
501                     *        BC preserved, AF corrupted
502                     *
503 CBA8 5E             RSVHL   MOV   E,M       ) Stringpointer
504 CBA9 3600                   MVI   M,:00     ) in DE and then
505 CBAB 23                     INX   H         ) erased.
506 CBAC 56                     MOV   D,M       )
507 CBAD 3600                   MVI   M,:00     )
508 CBAF 23                     INX   H
509 CBB0 E5                     PUSH  H
510 CBB1 2A9F02                 LHLD  :029F     Get startaddr textbuf
511 CBB4 EB                     XCHG            in DE; stringpntr in HL
512 CBB5 7C                     MOV   A,H
513 CBB6 B5                     ORA   L         Stringpntr is already 0 ?
514 CBB7 C414DE                 CNZ   :DE14     If not: test if end of
515                                             heap reached
516 CBBA DC87D1                 CC    :D187     If not: clear heap entry
517 CBBD E1                     POP   H
518 CBBE C9                     RET
519                     *
520                     *
521                     *
522 CBBF                        END
```

****
* SYMBOL TABLE *
****

ASKRM	CA01	ASR10	CA1B	ASR20	CA1E	ASR30	CA21
DADD	CAAE	DADR	CAB1	EAR10	CB8E	EAR20	CB98
EAR30	CB99	EARRAY	CB5B	EMSTP	CA25	FDL05	CAFF
FDL10	CB03	FDL20	CB1C	FDL30	CB1D	FINDL	CAF6
FNAME	CA95	FNM10	CA9B	FNM20	CAAA	LC188	CB56
LK10	CA5F	LK15	CA69	LK20	CA6F	LK30	CA78
LK40	CA8B	LK50	CA8F	LKC10	CA37	LKC20	CA3D
LKC30	CA4E	LKI10	CAE7	LOOK	CA57	LOOKC	CA34
LOOKI	CAB8	LOOKX	CA5A	RSVHL	CBA8	SCRATC	CB23
SCT10	CB2A	SCT30	CB47	SCT40	CB4D	SCT90	CB53
SMKRM	CA01	ZFPINT	CB9E				

```
002                             ORG   :CBBF
003                     *
004                     *
005                     *
006                     ****************************
007                     * STRINGS BASIC COMMANDS *
008                     ****************************
009                     *
010                     * The first byte of each string is a length byte.
011                     *
012                     * The first byte after the string is the 'type'
013                     * byte. It is used to compose the TOKEN of the
014                     * particular Basic command:
015                     *    type byte, ANI :3F, ORI :80, gives TOKEN.
016                     *
017                     * Commands with type bytes bit 7=1 can be executed
018                     * during a program run. If bit 6=1, commands are
019                     * valid as direct command.
020                     *
021                     * The address given is the location of the encoding
022                     * routine for this particular command. These
023                     * routines can be found in ROM bank 3.
024                     *
025                     CMDTB
026 CBBF 03             SNEW    DATA  :03
027 CBC0 4E                     DATA  :4E       N
028 CBC1 45                     DATA  :45       E
029 CBC2 57                     DATA  :57       W
030 CBC3 81                     DATA  :81
031 CBC4 69E3                   DBL   :E369
032                     *
033 CBC6 04             SCONT   DATA  :04
034 CBC7 43                     DATA  :43       C
035 CBC8 4F                     DATA  :4F       O
036 CBC9 4E                     DATA  :4E       N
037 CBCA 54                     DATA  :54       T
038 CBCB 82                     DATA  :82
039 CBCC 69E3                   DBL   :E369
040                     *
041 CBCE 04             SSTOP   DATA  :04
042 CBCF 53                     DATA  :53       S
043 CBD0 54                     DATA  :54       T
044 CBD1 4F                     DATA  :4F       O
045 CBD2 50                     DATA  :50       P
046 CBD3 43                     DATA  :43
047 CBD4 69E3                   DBL   :E369
048                     *
049 CBD6 03             SEND    DATA  :03
050 CBD7 45                     DATA  :45       E
051 CBD8 4E                     DATA  :4E       N
052 CBD9 44                     DATA  :44       D
053 CBDA 44                     DATA  :44
054 CBDB 69E3                   DBL   :E369
055                     *
056 CBDD 07             SREST   DATA  :07
057 CBDE 52                     DATA  :52       R
058 CBDF 45                     DATA  :45       E
059 CBE0 53                     DATA  :53       S
060 CBE1 54                     DATA  :54       T
061 CBE2 4F                     DATA  :4F       O
062 CBE3 52                     DATA  :52       R
063 CBE4 45                     DATA  :45       E
```

```
064 CBE5 C5                     DATA  :C5
065 CBE6 69E3                   DBL   :E369
066                     *
067 CBE8 06             SRET    DATA  :06
068 CBE9 52                     DATA  :52       R
069 CBEA 45                     DATA  :45       E
070 CBEB 54                     DATA  :54       T
071 CBEC 55                     DATA  :55       U
072 CBED 52                     DATA  :52       R
073 CBEE 4E                     DATA  :4E       N
074 CBEF 46                     DATA  :46
075 CBF0 69E3                   DBL   :E369
076                     *
077 CBF2 03             SRUN    DATA  :03
078 CBF3 52                     DATA  :52       R
079 CBF4 55                     DATA  :55       U
080 CBF5 4E                     DATA  :4E       N
081 CBF6 87                     DATA  :87
082 CBF7 95E2                   DBL   :E295
083                     *
084 CBF9 04             SGOTO   DATA  :04
085 CBFA 47                     DATA  :47       G
086 CBFB 4F                     DATA  :4F       O
087 CBFC 54                     DATA  :54       T
088 CBFD 4F                     DATA  :4F       O
089 CBFE 49                     DATA  :49
090 CBFF 6AE3                   DBL   :E36A
091                     *
092 CC01 05             SGOSUB  DATA  :05
093 CC02 47                     DATA  :47       G
094 CC03 4F                     DATA  :4F       O
095 CC04 53                     DATA  :53       S
096 CC05 55                     DATA  :55       U
097 CC06 42                     DATA  :42       B
098 CC07 4A                     DATA  :4A
099 CC08 6AE3                   DBL   :E36A
100                     *
101 CC0A 03             SIMP    DATA  :03
102 CC0B 49                     DATA  :49       I
103 CC0C 4D                     DATA  :4D       M
104 CC0D 50                     DATA  :50       P
105 CC0E B5                     DATA  :B5
106 CC0F 9FE2                   DBL   :E29F
107                     *
108 CC11 05             SSAVA   DATA  :05
109 CC12 53                     DATA  :53       S
110 CC13 41                     DATA  :41       A
111 CC14 56                     DATA  :56       V
112 CC15 45                     DATA  :45       E
113 CC16 41                     DATA  :41       A
114 CC17 F9                     DATA  :F9
115 CC18 A9E4                   DBL   :E4A9
116                     *
117 CC1A 05             SLODA   DATA  :05
118 CC1B 4C                     DATA  :4C       L
119 CC1C 4F                     DATA  :4F       O
120 CC1D 41                     DATA  :41       A
121 CC1E 44                     DATA  :44       D
122 CC1F 41                     DATA  :41       A
123 CC20 FA                     DATA  :FA
124 CC21 A9E4                   DBL   :E4A9
125                     *
```

```
126 CC23 03          SOUT    DATA  :03
127 CC24 4F                  DATA  :4F       O
128 CC25 55                  DATA  :55       U
129 CC26 54                  DATA  :54       T
130 CC27 CE                  DATA  :CE
131 CC28 02E3                DBL   :E302
132                  *
133 CC2A 04          SPOKE   DATA  :04
134 CC2B 50                  DATA  :50       P
135 CC2C 4F                  DATA  :4F       O
136 CC2D 4B                  DATA  :4B       K
137 CC2E 45                  DATA  :45       E
138 CC2F CF                  DATA  :CF
139 CC30 02E3                DBL   :E302
140                  *
141 CC32 04          SWAIT   DATA  :04
142 CC33 57                  DATA  :57       W
143 CC34 41                  DATA  :41       A
144 CC35 49                  DATA  :49       I
145 CC36 54                  DATA  :54       T
146 CC37 D0                  DATA  :D0
147 CC38 59E2                DBL   :E259
148                  *
149 CC3A 04          SLIST   DATA  :04
150 CC3B 4C                  DATA  :4C       L
151 CC3C 49                  DATA  :49       I
152 CC3D 53                  DATA  :53       S
153 CC3E 54                  DATA  :54       T
154 CC3F D3                  DATA  :D3
155 CC40 28E2                DBL   :E228
156                  *
157 CC42 04          SEDIT   DATA  :04
158 CC43 45                  DATA  :45       E
159 CC44 44                  DATA  :44       D
160 CC45 49                  DATA  :49       I
161 CC46 54                  DATA  :54       T
162 CC47 B6                  DATA  :B6
163 CC48 28E2                DBL   :E228
164                  *
165 CC4A 05          SSOUND  DATA  :05
166 CC4B 53                  DATA  :53       S
167 CC4C 4F                  DATA  :4F       O
168 CC4D 55                  DATA  :55       U
169 CC4E 4E                  DATA  :4E       N
170 CC4F 44                  DATA  :44       D
171 CC50 D6                  DATA  :D6
172 CC51 17E3                DBL   :E317
173                  *
174 CC53 05          SNOISE  DATA  :05
175 CC54 4E                  DATA  :4E       N
176 CC55 4F                  DATA  :4F       O
177 CC56 49                  DATA  :49       I
178 CC57 53                  DATA  :53       S
179 CC58 45                  DATA  :45       E
180 CC59 D7                  DATA  :D7
181 CC5A 25E3                DBL   :E325
182                  *
183 CC5C 08          SENV    DATA  :08
184 CC5D 45                  DATA  :45       E
185 CC5E 4E                  DATA  :4E       N
186 CC5F 56                  DATA  :56       V
187 CC60 45                  DATA  :45       E
```

```
188 CC61 4C                     DATA  :4C       L
189 CC62 4F                     DATA  :4F       O
190 CC63 50                     DATA  :50       P
191 CC64 45                     DATA  :45       E
192 CC65 D8                     DATA  :D8
193 CC66 F6E1                   DBL   :E1F6
194                     *
195 CC68 06             SCURS   DATA  :06
196 CC69 43                     DATA  :43       C
197 CC6A 55                     DATA  :55       U
198 CC6B 52                     DATA  :52       R
199 CC6C 53                     DATA  :53       S
200 CC6D 4F                     DATA  :4F       O
201 CC6E 52                     DATA  :52       R
202 CC6F D9                     DATA  :D9
203 CC70 02E3                   DBL   :E302
204                     *
205 CC72 04             SMODE   DATA  :04
206 CC73 4D                     DATA  :4D       M
207 CC74 4F                     DATA  :4F       O
208 CC75 44                     DATA  :44       D
209 CC76 45                     DATA  :45       E
210 CC77 DA                     DATA  :DA
211 CC78 D1E1                   DBL   :E1D1
212                     *
213 CC7A 03             SDOT    DATA  :03
214 CC7B 44                     DATA  :44       D
215 CC7C 4F                     DATA  :4F       O
216 CC7D 54                     DATA  :54       T
217 CC7E DB                     DATA  :DB
218 CC7F 8FE2                   DBL   :E28F
219                     *
220 CC81 04             SDRAW   DATA  :04
221 CC82 44                     DATA  :44       D
222 CC83 52                     DATA  :52       R
223 CC84 41                     DATA  :41       A
224 CC85 57                     DATA  :57       W
225 CC86 DC                     DATA  :DC
226 CC87 8CE2                   DBL   :E28C
227                     *
228 CC89 04             SFILL   DATA  :04
229 CC8A 46                     DATA  :46       F
230 CC8B 49                     DATA  :49       I
231 CC8C 4C                     DATA  :4C       L
232 CC8D 4C                     DATA  :4C       L
233 CC8E DD                     DATA  :DD
234 CC8F 8CE2                   DBL   :E28C
235                     *
236 CC91 06             SCOLT   DATA  :06
237 CC92 43                     DATA  :43       C
238 CC93 4F                     DATA  :4F       O
239 CC94 4C                     DATA  :4C       L
240 CC95 4F                     DATA  :4F       O
241 CC96 52                     DATA  :52       R
242 CC97 54                     DATA  :54       T
243 CC98 DE                     DATA  :DE
244 CC99 0BE3                   DBL   :E30B
245                     *
246 CC9B 06             SCOLG   DATA  :06
247 CC9C 43                     DATA  :43       C
248 CC9D 4F                     DATA  :4F       O
249 CC9E 4C                     DATA  :4C       L
```

```
250 CC9F 4F                 DATA  :4F       O
251 CCA0 52                 DATA  :52       R
252 CCA1 47                 DATA  :47       G
253 CCA2 DF                 DATA  :DF
254 CCA3 0BE3               DBL   :E30B
255                 *
256 CCA5 05         SINPUT  DATA  :05
257 CCA6 49                 DATA  :49       I
258 CCA7 4E                 DATA  :4E       N
259 CCA8 50                 DATA  :50       P
260 CCA9 55                 DATA  :55       U
261 CCAA 54                 DATA  :54       T
262 CCAB 60                 DATA  :60
263 CCAC 15E1               DBL   :E115
264                 *
265 CCAE 04         SDATA   DATA  :04
266 CCAF 44                 DATA  :44       D
267 CCB0 41                 DATA  :41       A
268 CCB1 54                 DATA  :54       T
269 CCB2 41                 DATA  :41       A
270 CCB3 62                 DATA  :62
271 CCB4 A2E8               DBL   :E8A2
272                 *
273 CCB6 04         SREAD   DATA  :04
274 CCB7 52                 DATA  :52       R
275 CCB8 45                 DATA  :45       E
276 CCB9 41                 DATA  :41       A
277 CCBA 44                 DATA  :44       D
278 CCBB 63                 DATA  :63
279 CCBC 27E1               DBL   :E127
280                 *
281 CCBE 03         SLET    DATA  :03
282 CCBF 4C                 DATA  :4C       L
283 CCC0 45                 DATA  :45       E
284 CCC1 54                 DATA  :54       T
285 CCC2 E4                 DATA  :E4
286 CCC3 FEE0               DBL   :E0FE
287                 *
288 CCC5 02         SIF     DATA  :02
289 CCC6 49                 DATA  :49       I
290 CCC7 46                 DATA  :46       F
291 CCC8 66                 DATA  :66
292 CCC9 BCE0               DBL   :E0BC
293                 *
294 CCCB 03         SREM    DATA  :03
295 CCCC 52                 DATA  :52       R
296 CCCD 45                 DATA  :45       E
297 CCCE 4D                 DATA  :4D       M
298 CCCF 69                 DATA  :69
299 CCD0 66E3               DBL   :E366
300                 *
301 CCD2 03         SFOR    DATA  :03
302 CCD3 46                 DATA  :46       F
303 CCD4 4F                 DATA  :4F       O
304 CCD5 52                 DATA  :52       R
305 CCD6 EA                 DATA  :EA
306 CCD7 5FE0               DBL   :E05F
307                 *
308 CCD9 04         SNEXT   DATA  :04
309 CCDA 4E                 DATA  :4E       N
310 CCDB 45                 DATA  :45       E
311 CCDC 58                 DATA  :58       X
```

```
312 CCDD 54                     DATA  :54       T
313 CCDE EB                     DATA  :EB
314 CCDF A9E0                   DBL   :E0A9
315                     *
316 CCE1 05             SPRINT  DATA  :05
317 CCE2 50                     DATA  :50       P
318 CCE3 52                     DATA  :52       R
319 CCE4 49                     DATA  :49       I
320 CCE5 4E                     DATA  :4E       N
321 CCE6 54                     DATA  :54       T
322 CCE7 ED                     DATA  :ED
323 CCE8 9FE1                   DBL   :E19F
324                     *
325 CCEA 01             S?      DATA  :01
326 CCEB 3F                     DATA  :3F       ?  (abbr. for PRINT)
327 CCEC ED                     DATA  :ED
328 CCED 9FE1                   DBL   :E19F
329                     *
330 CCEF 02             SON     DATA  :02
331 CCF0 4F                     DATA  :4F       O
332 CCF1 4E                     DATA  :4E       N
333 CCF2 6E                     DATA  :6E
334 CCF3 76E1                   DBL   :E176
335                     *
336 CCF5 03             SDIM    DATA  :03
337 CCF6 44                     DATA  :44       D
338 CCF7 49                     DATA  :49       I
339 CCF8 4D                     DATA  :4D       M
340 CCF9 F0                     DATA  :F0
341 CCFA 66E1                   DBL   :E166
342                     *
343 CCFC 03                     DATA  :03
344 CCFD 2A                     DATA  :2A       *
345 CCFE 2A                     DATA  :2A       *
346 CCFF 2A                     DATA  :2A       *
347 CD00 71                     DATA  :71
348 CD01 66E3                   DBL   :E366
349                     *
350 CD03 02             SUT     DATA  :02
351 CD04 55                     DATA  :55       U
352 CD05 54                     DATA  :54       T
353 CD06 B2                     DATA  :B2
354 CD07 69E3                   DBL   :E369
355                     *
356 CD09 05             SCALM   DATA  :05
357 CD0A 43                     DATA  :43       C
358 CD0B 41                     DATA  :41       A
359 CD0C 4C                     DATA  :4C       L
360 CD0D 4C                     DATA  :4C       L
361 CD0E 4D                     DATA  :4D       M
362 CD0F F3                     DATA  :F3
363 CD10 44E3                   DBL   :E344
364                     *
365 CD12 05             SCLEAR  DATA  :05
366 CD13 43                     DATA  :43       C
367 CD14 4C                     DATA  :4C       L
368 CD15 45                     DATA  :45       E
369 CD16 41                     DATA  :41       A
370 CD17 52                     DATA  :52       R
371 CD18 F4                     DATA  :F4
372 CD19 14E3                   DBL   :E314
373                     *
```

```
374 CD1B 04          SLOAD   DATA  :04
375 CD1C 4C                  DATA  :4C       L
376 CD1D 4F                  DATA  :4F       O
377 CD1E 41                  DATA  :41       A
378 CD1F 44                  DATA  :44       D
379 CD20 CB                  DATA  :CB
380 CD21 55E3                DBL   :E355
381                  *
382 CD23 04          SSAVE   DATA  :04
383 CD24 53                  DATA  :53       S
384 CD25 41                  DATA  :41       A
385 CD26 56                  DATA  :56       V
386 CD27 45                  DATA  :45       E
387 CD28 8C                  DATA  :8C
388 CD29 55E3                DBL   :E355
389                  *
390 CD2B 05          SCHECK  DATA  :05
391 CD2C 43                  DATA  :43       C
392 CD2D 48                  DATA  :48       H
393 CD2E 45                  DATA  :45       E
394 CD2F 43                  DATA  :43       C
395 CD30 4B                  DATA  :4B       K
396 CD31 8D                  DATA  :8D
397 CD32 69E3                DBL   :E369
398                  *
399 CD34 05                  DATA  :05       (Cancelled instr.)
400 CD35 00                  DATA  :00
401 CD36 52                  DATA  :52       R
402 CD37 41                  DATA  :41       A
403 CD38 53                  DATA  :53       S
404 CD39 45                  DATA  :45       E
405 CD3A FB                  DATA  :FB
406 CD3B 73E8                DBL   :E873
407                  *
408 CD3D 04          SSTEP   DATA  :04
409 CD3E 53                  DATA  :53       S
410 CD3F 54                  DATA  :54       T
411 CD40 45                  DATA  :45       E
412 CD41 50                  DATA  :50       P
413 CD42 BC                  DATA  :BC
414 CD43 69E3                DBL   :E369
415                  *
416 CD45 04          STRON   DATA  :04
417 CD46 54                  DATA  :54       T
418 CD47 52                  DATA  :52       R
419 CD48 4F                  DATA  :4F       O
420 CD49 4E                  DATA  :4E       N
421 CD4A FD                  DATA  :FD
422 CD4B 69E3                DBL   :E369
423                  *
424 CD4D 05          STROF   DATA  :05
425 CD4E 54                  DATA  :54       T
426 CD4F 52                  DATA  :52       R
427 CD50 4F                  DATA  :4F       O
428 CD51 46                  DATA  :46       F
429 CD52 46                  DATA  :46       F
430 CD53 FE                  DATA  :FE
431 CD54 69E3                DBL   :E369
432                  *
433 CD56 04          STALK   DATA  :04
434 CD57 54                  DATA  :54       T
435 CD58 41                  DATA  :41       A
```

```
436 CD59 4C                    DATA  :4C       L
437 CD5A 4B                    DATA  :4B       K
438 CD5B FB                    DATA  :FB
439 CD5C 14E3                  DBL   :E314
440                    *
441 CD5E 00                    DATA  :00       End of table
442 CD5F E5                    DATA  :E5       Assignment (= LET)
443 CD60 FEE0                  DBL   :E0FE
444                    *
445 CD62 FF                    DATA  :FF
446 CD63 FF                    DATA  :FF
447                    *
448                    **********************
449                    * RUN basiccmd TALK *
450                    **********************
451                    *
452 CD64 CDF8E6        RTALK   CALL  :E6F8     (0) Get addr parameter
453                                            block in HL
454 CD67 7E            RTK10   MOV   A,M       Get code
455 CD68 23                    INX   H
456 CD69 B7                    ORA   A
457 CD6A F8                    RM              Ready if code = FF (end)
458 CD6B FE05                  CPI   :05
459 CD6D DA45CE                JC    :CE45     Jump if freq. code
460 CD70 FE0C                  CPI   :0C
461 CD72 DA94EE                JC    :EE94     (0) Jump if volume code
462 CD75 C36DEC                JMP   :EC6D     (0) Continu
463                    *
464                    ****************
465                    * STRING DATA *
466                    ****************
467                    *
468 CD78 08            LC236   DATA  :08
469 CD79 57                    DATA  :57       W
470 CD7A 41                    DATA  :41       A
471 CD7B 49                    DATA  :49       I
472 CD7C 54                    DATA  :54       T
473 CD7D 20                    DATA  :20
474 CD7E 4D                    DATA  :4D       M
475 CD7F 45                    DATA  :45       E
476 CD80 4D                    DATA  :4D       M
477                    *
478 CD81 09            LC237   DATA  :09
479 CD82 57                    DATA  :57       W
480 CD83 41                    DATA  :41       A
481 CD84 49                    DATA  :49       I
482 CD85 54                    DATA  :54       T
483 CD86 20                    DATA  :20
484 CD87 54                    DATA  :54       T
485 CD88 49                    DATA  :49       I
486 CD89 4D                    DATA  :4D       M
487 CD8A 45                    DATA  :45       E
488                    *
489                    *
490                    *
491 CD8B                       END

****************************
* S Y M B O L   T A B L E *
****************************

CMDTB  CBBF    LC236  CD78    LC237  CD81    RTALK  CD64
```

RTK10	CD67	S?	CCEA	SCALM	CD09	SCHECK	CD2B
SCLEAR	CD12	SCOLG	CC9B	SCOLT	CC91	SCONT	CBC6
SCURS	CC68	SDATA	CCAE	SDIM	CCF5	SDOT	CC7A
SDRAW	CC81	SEDIT	CC42	SEND	CBD6	SENV	CC5C
SFILL	CC89	SFOR	CCD2	SGOSUB	CC01	SGOTO	CBF9
SIF	CCC5	SIMP	CC0A	SINPUT	CCA5	SLET	CCBE
SLIST	CC3A	SLOAD	CD1B	SLODA	CC1A	SMODE	CC72
SNEW	CBBF	SNEXT	CCD9	SNOISE	CC53	SON	CCEF
SOUT	CC23	SPOKE	CC2A	SPRINT	CCE1	SREAD	CCB6
SREM	CCCB	SREST	CBDD	SRET	CBE8	SRUN	CBF2
SSAVA	CC11	SSAVE	CD23	SSOUND	CC4A	SSTEP	CD3D
SSTOP	CBCE	STALK	CD56	STROF	CD4D	STRON	CD45
SUT	CD03	SWAIT	CC32				

```
002                             ORG   :CD8B
003                     *
004                     *
005                     *
006                     ****************************************
007                     * POINTERS TO STRINGS OF BASIC COMMANDS *
008                     ****************************************
009                     *
010                     * This table, with base at CC08, is used for
011                     * printing the Basic instructions during a
012                     * listing.
013                     *
014                     * From the TOKEN, the address in this table can
015                     * be found by:
016                     *       CC08 + 3x TOKEN = address in table
017                     * This is done by a routine on OECCC.
018                     *
019                     * The address given points to the memory location
020                     * on which the particular string can be found.
021                     *
022                     * The data byte after the address is an offset
023                     * with base at OECF8. Therewith the instructions
024                     * can be found about what to print after the
025                     * Basic statement (when performing LIST).
026                     *
027 CD8B BFCB        CDTAB   DBL   :CBBF     NEW
028 CD8D 00                  DATA  :00
029                     *
030 CD8E C6CB                DBL   :CBC6     CONT
031 CD90 00                  DATA  :00
032                     *
033 CD91 CECB                DBL   :CBCE     STOP
034 CD93 00                  DATA  :00
035                     *
036 CD94 D6CB                DBL   :CBD6     END
037 CD96 00                  DATA  :00
038                     *
039 CD97 DDCB                DBL   :CBDD     RESTORE
040 CD99 00                  DATA  :00
041                     *
042 CD9A E8CB                DBL   :CBE8     RETURN
043 CD9C 00                  DATA  :00
044                     *
045 CD9D F2CB                DBL   :CBF2     RUN
046 CD9F 00                  DATA  :00
047                     *
048 CDA0 F2CB                DBL   :CBF2     RUN
049 CDA2 01                  DATA  :01
050                     *
051 CDA3 F9CB                DBL   :CBF9     GOTO
052 CDA5 01                  DATA  :01
053                     *
054 CDA6 01CC                DBL   :CC01     GOSUB
055 CDA8 01                  DATA  :01
056                     *
057 CDA9 1BCD                DBL   :CD1B     LOAD
058 CDAB 04                  DATA  :04
059                     *
060 CDAC 23CD                DBL   :CD23     SAVE
061 CDAE 04                  DATA  :04
062                     *
063 CDAF 2BCD                DBL   :CD2B     CHECK
```

```
064 CDB1 00                    DATA  :00
065                  *
066 CDB2 23CC                  DBL   :CC23     OUT
067 CDB4 05                    DATA  :05
068                  *
069 CDB5 2ACC                  DBL   :CC2A     POKE
070 CDB7 05                    DATA  :05
071                  *
072 CDB8 32CC                  DBL   :CC32     WAIT
073 CDBA 0A                    DATA  :0A
074                  *
075 CDBB 78CD                  DBL   :CD78     WAIT MEM
076 CDBD 0A                    DATA  :0A
077                  *
078 CDBE 81CD                  DBL   :CD81     WAIT TIME
079 CDC0 04                    DATA  :04
080                  *
081 CDC1 3ACC                  DBL   :CC3A     LIST
082 CDC3 00                    DATA  :00
083                  *
084 CDC4 3ACC                  DBL   :CC3A     LIST
085 CDC6 01                    DATA  :01
086                  *
087 CDC7 3ACC                  DBL   :CC3A     LIST
088 CDC9 0B                    DATA  :0B
089                  *
090 CDCA 4ACC                  DBL   :CC4A     SOUND
091 CDCC 0C                    DATA  :0C
092                  *
093 CDCD 53CC                  DBL   :CC53     NOISE
094 CDCF 0D                    DATA  :0D
095                  *
096 CDD0 5CCC                  DBL   :CC5C     ENVELOPE
097 CDD2 0E                    DATA  :0E
098                  *
099 CDD3 68CC                  DBL   :CC68     CURSOR
100 CDD5 05                    DATA  :05
101                  *
102 CDD6 75CC                  DBL   :CC75     MODE
103 CDD8 0F                    DATA  :0F
104                  *
105 CDD9 7ACC                  DBL   :CC7A     DOT
106 CDDB 07                    DATA  :07
107                  *
108 CDDC 81CC                  DBL   :CC81     DRAW
109 CDDE 08                    DATA  :08
110                  *
111 CDDF 89CC                  DBL   :CC89     FILL
112 CDE1 08                    DATA  :08
113                  *
114 CDE2 91CC                  DBL   :CC91     COLORT
115 CDE4 09                    DATA  :09
116                  *
117 CDE5 9BCC                  DBL   :CC9B     COLORG
118 CDE7 09                    DATA  :09
119                  *
120 CDE8 A5CC                  DBL   :CCA5     INPUT
121 CDEA 11                    DATA  :11
122                  *
123 CDEB A5CC                  DBL   :CCA5     INPUT
124 CDED 10                    DATA  :10
125                  *
```

```
126 CDEE AECC                DBL   :CCAE     DATA
127 CDF0 03                  DATA  :03
128                  *
129 CDF1 B6CC                DBL   :CCB6     READ
130 CDF3 11                  DATA  :11
131                  *
132 CDF4 BECC                DBL   :CCBE     LET
133 CDF6 13                  DATA  :13
134                  *
135 CDF7 0000                DBL   :0000     Assignment (= LET)
136 CDF9 13                  DATA  :13
137                  *
138 CDFA C5CC                DBL   :CCC5     IF
139 CDFC 14                  DATA  :14
140                  *
141 CDFD C5CC                DBL   :CCC5     IF
142 CDFF 15                  DATA  :15
143                  *
144 CE00 C5CC                DBL   :CCC5     IF
145 CE02 16                  DATA  :16
146                  *
147 CE03 CBCC                DBL   :CCCB     REM
148 CE05 03                  DATA  :03
149                  *
150 CE06 D2CC                DBL   :CCD2     FOR
151 CE08 17                  DATA  :17
152                  *
153 CE09 D9CC                DBL   :CCD9     NEXT
154 CE0B 00                  DATA  :00
155                  *
156 CE0C D9CC                DBL   :CCD9     NEXT
157 CE0E 18                  DATA  :18
158                  *
159 CE0F E1CC                DBL   :CCE1     PRINT
160 CE11 19                  DATA  :19
161                  *
162 CE12 EFCC                DBL   :CCEF     ON
163 CE14 1A                  DATA  :1A
164                  *
165 CE15 EFCC                DBL   :CCEF     ON
166 CE17 1B                  DATA  :1B
167                  *
168 CE18 F5CC                DBL   :CCF5     DIM
169 CE1A 11                  DATA  :11
170                  *
171 CE1B FCCC                DBL   :CCFC     '***'
172 CE1D 03                  DATA  :03
173                  *
174 CE1E 03CD                DBL   :CD03     UT
175 CE20 00                  DATA  :00
176                  *
177 CE21 09CD                DBL   :CD09     CALLM
178 CE23 1C                  DATA  :1C
179                  *
180 CE24 12CD                DBL   :CD12     CLEAR
181 CE26 04                  DATA  :04
182                  *
183 CE27 0000                DBL   :0000     IMP
184 CE29 00                  DATA  :00
185                  *
186 CE2A 3ACC                DBL   :CC3A     LIST
187 CE2C 00                  DATA  :00
```

```
188                     *
189 CE2D 3ACC                  DBL   :CC3A     LIST
190 CE2F 01                    DATA  :01
191                     *
192 CE30 3ACC                  DBL   :CC3A     LIST
193 CE32 0B                    DATA  :0B
194                     *
195 CE33 11CC                  DBL   :CC11     SAVEA
196 CE35 1E                    DATA  :1E
197                     *
198 CE36 1ACC                  DBL   :CC1A     LOADA
199 CE38 1E                    DATA  :1E
200                     *
201 CE39 56CD                  DBL   :CD56     TALK
202 CE3B 04                    DATA  :04
203                     *
204 CE3C 3DCD                  DBL   :CD3D     STEP
205 CE3E 00                    DATA  :00
206                     *
207 CE3F 45CD                  DBL   :CD45     TRON
208 CE41 00                    DATA  :00
209                     *
210 CE42 4DCD                  DBL   :CD4D     TROFF
211 CE44 00                    DATA  :00
212                     *
213                     ******************************
214                     * part of Run 'TALK' (CD6D) *
215                     ******************************
216                     *
217                     * Set frequencies of channels 0,1 or 2.
218                     *
219 CE45 5F             RTK60   MOV   E,A       Code in E (=lobyte
220                                             osc. addr)
221 CE46 16FC                  MVI   D,:FC
222 CE48 7E                    MOV   A,M       Get 1st byte freq.code
223 CE49 12                    STAX  D         into osc.
224 CE4A 23                    INX   H
225 CE4B 7E                    MOV   A,M       Get 2nd byte freq.code
226 CE4C 12                    STAX  D         into osc.
227 CE4D C347EA                JMP   :EA47     (0) Handle next code
228                     *
229 CE50 FF                    DATA  :FF
230                     *
231                     ******************************
232                     * GET (M+1) IN E, ZERO M+1 *
233                     ******************************
234                     *
235                     * Part of EARRAY (CB5B).
236                     *
237                     * Entry: HL points to M.
238                     * Exit:  HL points to M+1. (M+1) in E.
239                     *        AFBCD preserved.
240                     *
241 CE51 23             MPT09   INX   H
242 CE52 5E                    MOV   E,M
243 CE53 3600                  MVI   M,:00
244 CE55 C9                    RET
245                     *
246                     ***************
247                     * STRING DATA *
248                     ***************
249                     *
```

```
CE56 05          MSPACE  DATA  :05
CE57 53                  DATA  :53        S
CE58 50                  DATA  :50        P
CE59 41                  DATA  :41        A
CE5A 43                  DATA  :43        C
CE5B 45                  DATA  :45        E
                 *
                 ****************************
                 * part of RUN DIM (OE639) *
                 ****************************
                 *
CE5C 2B          MPT41   DCX   H
CE5D C35BCB              JMP   :CB5B      Erase array if exists
                 *
                 ************************************
                 * GET TABNUMBER IN L, DOUTC IN A *
                 ************************************
                 *
                 * Part of Run 'TAB'.
                 *
CE60 CD1DE7      MPT50   CALL  :E71D      (0) Get nr of tabs in A
CE63 6F                  MOV   L,A       save it in L
CE64 3A3101              LDA   :0131     Get output direction
CE67 C9                  RET
                 *
                 ******************************************
                 * PRINT EXPRESSION FOLLOWED BY A SPACE *
                 ******************************************
                 *
                 * Entry SCHSP frequently used to print a space.
                 *
CE68 CDA2EE      LC230   CALL  :EEA2      (0) Print expression
CE6B 3E20        SCHSP   MVI   A,:20
CE6D C360DD              JMP   :DD60     Print space
                 *
                 *************
                 * PRINT ',' *
                 *************
                 *
                 * Entry: None.
                 * Exit:  FBCDEHL preserved.
                 *
CE70 3E2C        SCHCO   MVI   A,:2C
CE72 C360DD              JMP   :DD60     Print ','
                 *
                 ***********************************
                 * PRINT A STRING BETWEEN SPACES *
                 ***********************************
                 *
                 * Entry: Pointer to stringpointer on stack.
                 * Exit:  BC preserved. AFDEHL corrupted.
                 *
CE75 CD6BCE      STXSS   CALL  :CE6B      Print space
CE78 E3          STXTS   XTHL             Get stringpntr from stack
CE79 5E                  MOV   E,M       ) Store addr string in DE
CE7A 23                  INX   H         )
CE7B 56                  MOV   D,M       )
CE7C 23                  INX   H         )
CE7D E3                  XTHL            Addr after pntr on stack
CE7E EB                  XCHG            Addr string in HL
CE7F CD32DB              CALL  :DB32     Print string pointed by HL
CE82 C36BCE              JMP   :CE6B     Print space
```

```
312                     *
313                     ******************************
314                     * EDIT: PRINT TEXT COMPLETE *
315                     ******************************
316                     *
317 CE85 CD17EF         LC216   CALL  :EF17     (2) Print text complete
318 CE88 C338E1                 JMP   :E138     (2) Popall, ret
319                     *
320                     ***********************************
321                     * LIST ARRAY NAME - (not used) *
322                     ***********************************
323                     *
324 CE8B D5             LC217   PUSH  D
325 CE8C CDF7EE                 CALL  :EEF7     (0) List array name
326 CE8F D1                     POP   D
327 CE90 C9                     RET
328                     *
329                     ****************
330                     * part of C6BA *
331                     ****************
332                     *
333 CE91 E5             LC196   PUSH  H
334 CE92 C32DE9                 JMP   :E92D     (2) Now set up screen bits
335                                             for mode 1
336                     *
337                     **************
338                     * (not used) *
339                     **************
340                     *
341 CE95 3AA200         LC218   LDA   :00A2     Get startaddr edit buffer
342 CE98 C37CE9                 JMP   :E97C     (0)
343                     *
344                     ****************************
345                     * CONVERT MACC FOR OUTPUT *
346                     ****************************
347                     *
348                     * The MACC contents is converted from FPT to ASCII.
349                     *
350                     * Exit: AF corrupted, BCDEHL preserved.
351                     *
352 CE9B CD21C0         FBCP    CALL  :C021     Convert FPT nr for output
353 CE9E C5                     PUSH  B
354 CE9F 0601                   MVI   B,:01     Cannot trim last dec.place
355 CEA1 C365DB                 JMP   :DB65     Tidy up into external form
356                     *
357                     **********************
358                     * LIST CURRENT LINE *
359                     **********************
360                     *
361                     * Lists a program line if trace flag set.
362                     * Part of C8F5.
363                     *
364 CEA4 0B             MPT06   DCX   B
365 CEA5 CD55DD                 CALL  :DD55     Cursor to begin next line
366 CEA8 C3ABEC                 JMP   :ECAB     (0) List current line
367                     *
368 CEAB D1             LC219   POP   D         (Not used)
369 CEAC C9                     RET
370                     *
371                     ************************
372                     * part of SMKRM (CA01) *
373                     ************************
```

```
374                     *
375 CEAD D1             MPT07   POP   D          Return, CY=1
376 CEAE F1                     POP   PSW
377 CEAF 37                     STC
378 CEB0 C9                     RET
379                     *
380 CEB1 F1             MPT08   POP   PSW        Return, CY=0
381 CEB2 37                     STC
382 CEB3 3F                     CMC
383 CEB4 C9                     RET
384                     *
385                     ************************
386                     * CHANGE SCREEN MODE *
387                     ************************
388                     *
389                     * Part of Run 'MODE' (0E5BB).
390                     *
391                     * Entry: New mode in A.
392                     *
393 CEB5 EF             MPT40   RST   5          Change mode
394 CEB6 18                     DATA  :18
395 CEB7 DA10DA                 JC    :DA10      If insufficient memory:
396                                              error 'OUT OF MEMORY'.
397 CEBA C9                     RET
398                     *
399                     ********************************
400                     * part of RUN CLEAR (0E6B5) *
401                     ********************************
402                     *
403                     * Checks if more than 4 bytes are cleared.
404                     *
405                     * Entry: HL: Number of bytes to be cleared.
406                     *        F : flags on hibyte HL.
407                     *
408 CEBB 110400           MPT43   LXI   D,:0004    Must be at least 4 bytes
409 CEBE F414DE                   CP    :DE14      Compare HL-DE if not >32k
410 CEC1 DA15DA                   JC    :DA15      Run error 'NUMBER OUT
411                                              OF RANGE'if < 4.
412 CEC4 C9                     RET
413                     *
414 CEC5 FF                     DATA  :FF
415                     *
416                     ************************************
417                     * SET HEAP SIZE TO DEFAULT VALUE *
418                     ************************************
419                     *
420                     * Part of emergency stop routine (CA25).
421                     * Also runs a NEW command.
422                     *
423 CEC6 210001           HRNEW   LXI   H,:0100
424 CEC9 229D02                   SHLD  :029D      Store HEAP default value
425 CECC C3B5DE                   JMP   :DEB5      Run 'NEW'
426                     *
427                     ****************************************
428                     * RE-ORGANISE SCREEN AFTER 'DAI-PC' *
429                     ****************************************
430                     *
431                     * Lines after 'DAI PERSONAL COMPUTER' are set in
432                     * unit colour mode.
433                     * Set line mode byte to 7F and first char.
434                     * byte to 20, colour 00 during screen init.
435                     *
```

```
436                     * Entry: HL line mode byte of part. line.
437                     *
438 CECF 367F           MPT04   MVI   M,:7F     Wide char line contr byte
439 CED1 2B                     DCX   H
440 CED2 2B                     DCX   H
441 CED3 3620                   MVI   M,:20     Load space
442 CED5 2B                     DCX   H
443 CED6 3600                   MVI   M,:00     Colour no change
444 CED8 2B                     DCX   H
445 CED9 C9                     RET
446                     *
447                     ******************************************
448                     * part of RUN 'WAIT(TIME)' (DFD5/DFF7) *
449                     ******************************************
450                     *
451 CEDA 0B             REX1D   DCX   B
452 CEDB C31DE7                 JMP   :E71D     (0) Get value of argument
453                                             in A (max. FF)
454                     *
455                     ************************************************
456                     * EVALUATE ARGUMENTS IN NUMERIC EXPRESSION *
457                     ************************************************
458                     *
459                     * Not used.
460                     *
461 CEDE F5             LC231   PUSH  PSW
462 CEDF CD19E8                 CALL  :E819     (0) Evaluate arguments
463 CEE2 F1                     POP   PSW
464 CEE3 C9                     RET
465                     *
466                     ***************************************
467                     * SELECT ROM BANK 0; PRINT MESSAGE *
468                     ***************************************
469                     *
470                     * When CEE4 is called, the 2 bytes following
471                     * the CALL-instruction indicate the message to
472                     * be printed by the routine DAFF.
473                     *
474 CEE4 3A4000           SELB0   LDA   :0040     Get POROM
475 CEE7 E63F                     ANI   :3F       Select ROM bank 0
476 CEE9 324000                   STA   :0040     Set POROM
477 CEEC 3206FD                   STA   :FD06     and PORO
478 CEEF C3FFDA                   JMP   :DAFF     Print message
479                     *
480                     ********************************************
481                     * EDIT: RETURN FROM 'DELETE CHARACTER' *
482                     ********************************************
483                     *
484                     * Part of 2EFCC.
485                     *
486 CEF2 E1             LC232   POP   H
487 CEF3 CD30E3                 CALL  :E330     (2) Put cursor on screen
488 CEF6 C395EF                 JMP   :EF95     (2) Popall, ret, CY=1
489                     *
490                     ******************************************
491                     * PRINT 'COMPUTER' UNDER 'DAI PERSONAL' *
492                     ******************************************
493                     *
494                     * Part of RESET (C751).
495                     * During screen initialisation used to set a
496                     * new line mode byte between both parts of the
497                     * message. Line colour bytes are set for medium
```

```
498                     * resolution.
499                     *
500                     * Entry: HL: line mode byte to be changed.
501                     *         DE: offset for calculation next
502                     *             line mode byte.
503                     * Exit:   HL: next line mode byte.
504                     *
505 CEF9 365F            MPT03   MVI   M,:5F     Set line mode byte
506 CEFB 2B                      DCX   H
507 CEFC 3640                    MVI   M,:40     Set line colour byte
508 CEFE 23                      INX   H
509 CEFF 19                      DAD   D         Addr. next line mode byte
510 CF00 C9                      RET
511                     *
512 CF01 FF                      DATA  :FF
513                     *
514                     *
515                     *
516 CF02                         END
```

```
****************************
* S Y M B O L   T A B L E *
****************************
```

CDTAB	CD8B	FBCP	CE9B	HRNEW	CEC6	LC196	CE91
LC216	CE85	LC217	CE8B	LC218	CE95	LC219	CEAB
LC230	CE68	LC231	CEDE	LC232	CEF2	MPT03	CEF9
MPT04	CECF	MPT06	CEA4	MPT07	CEAD	MPT08	CEB1
MPT09	CE51	MPT40	CEB5	MPT41	CE5C	MPT43	CEBB
MPT50	CE60	MSPACE	CE56	REX1D	CEDA	RTK60	CE45
SCHCO	CE70	SCHSP	CE6B	SELBO	CEE4	STXSS	CE75
STXTS	CE78						

```
002                             ORG   :CF02
003                     *
004                     *
005                     *
006                     ****************************************
007                     * POINTERS TO ROUTINES BASICCOMMANDS *
008                     ****************************************
009                     *
010                     * This table, with base at CF00, gives the
011                     * addresses of the routines for execution of the
012                     * Basic statements.
013                     * The offset from baseaddress CF00 can be found
014                     * by adding 2x TOKEN to the base address.
015                     * Address indicates begin subroutine (ROM bank 0).
016                     *
017                     * The number given between brackets is the TOKEN.
018                     *
019 CF02 B5DE           CITAB   DBL   :DEB5     (81) NEW
020 CF04 D5DE                   DBL   :DED5     (82) CONT
021 CF06 03DF                   DBL   :DF03     (83) STOP
022 CF08 0CDF                   DBL   :DF0C     (84) END
023 CF0A 01E4                   DBL   :E401     (85) RESTORE
024 CF0C 4CDF                   DBL   :DF4C     (86) RETURN
025 CF0E 9EDF                   DBL   :DF9E     (87) RUN
026 CF10 BADF                   DBL   :DFBA     (88) RUN <linenumber>
027 CF12 63DF                   DBL   :DF63     (89) GOTO
028 CF14 2ADF                   DBL   :DF2A     (8A) GOSUB
029 CF16 70D2                   DBL   :D270     (8B) LOAD
030 CF18 3DD2                   DBL   :D23D     (8C) SAVE
031 CF1A C3D2                   DBL   :D2C3     (8D) CHECK
032 CF1C C9DF                   DBL   :DFC9     (8E) OUT
033 CF1E C0DF                   DBL   :DFC0     (8F) POKE
034 CF20 D5DF                   DBL   :DFD5     (90) WAIT
035 CF22 F7DF                   DBL   :DFF7     (91) WAIT MEM
036 CF24 16E0                   DBL   :E016     (92) WAIT TIME
037 CF26 97E1                   DBL   :E197     (93) LIST <whole program>
038 CF28 AAE1                   DBL   :E1AA     (94) LIST <linenumber>
039 CF2A B6E1                   DBL   :E1B6     (95) LIST <part of progr>
040 CF2C BCE4                   DBL   :E4BC     (96) SOUND
041 CF2E 0CE5                   DBL   :E50C     (97) NOISE
042 CF30 70E5                   DBL   :E570     (98) ENVELOPE
043 CF32 B2E5                   DBL   :E5B2     (99) CURSOR
044 CF34 BBE5                   DBL   :E5BB     (9A) MODE
045 CF36 C1E5                   DBL   :E5C1     (9B) DOT
046 CF38 CEE5                   DBL   :E5CE     (9C) DRAW
047 CF3A D7E5                   DBL   :E5D7     (9D) FILL
048 CF3C 0EE6                   DBL   :E60E     (9E) COLORT
049 CF3E 15E6                   DBL   :E615     (9F) COLORG
050 CF40 02E3                   DBL   :E302     (A0) INPUT
051 CF42 FCE2                   DBL   :E2FC     (A1) INPUT <with prompt>
052 CF44 8FE1                   DBL   :E18F     (A2) DATA
053 CF46 23E3                   DBL   :E323     (A3) READ
054 CF48 5AE4                   DBL   :E45A     (A4) LET
055 CF4A 5AE4                   DBL   :E45A     (A5) assignment
056 CF4C 20DF                   DBL   :DF20     (A6) IF THEN <statement>
057 CF4E 15DF                   DBL   :DF15     (A7) IF GOTO
058 CF50 15DF                   DBL   :DF15     (A8) IF THEN <linenumber>
059 CF52 8FE1                   DBL   :E18F     (A9) REM
060 CF54 2BE0                   DBL   :E02B     (AA) FOR .. TO
061 CF56 E5E0                   DBL   :E0E5     (AB) NEXT
062 CF58 C5E0                   DBL   :E0C5     (AC) NEXT <variable>
063 CF5A B3E2                   DBL   :E2B3     (AD) PRINT
```

```
064 CF5C 6ADF                DBL   :DF6A      (AE) ON GOTO
065 CF5E 71DF                DBL   :DF71      (AF) ON GOSUB
066 CF60 2FE6                DBL   :E62F      (B0) DIM
067 CF62 33DA                DBL   :DA33      (B1) *** (error line run)
068 CF64 9EE6                DBL   :E69E      (B2) UT
069 CF66 A4E6                DBL   :E6A4      (B3) CALLM
070 CF68 B5E6                DBL   :E6B5      (B4) CLEAR
071 CF6A 95E1                DBL   :E195      (B5) IMP
072 CF6C F5E1                DBL   :E1F5      (B6) EDIT <total>
073 CF6E 53E2                DBL   :E253      (B7) EDIT <linenumber>
074 CF70 5CE2                DBL   :E25C      (B8) EDIT <part>
075 CF72 1DD8                DBL   :D81D      (B9) SAVEA
076 CF74 5ED8                DBL   :D85E      (BA) LOADA
077 CF76 64CD                DBL   :CD64      (BB) TALK
078 CF78 FEDE                DBL   :DEFE      (BC) STEP
079 CF7A CEE6                DBL   :E6CE      (BD) TRON
080 CF7C D5E6                DBL   :E6D5      (BE) TROFF
081                     *
082                     ******************************
083                     * part of RUN 'ASC' (0EACF) *
084                     ******************************
085                     *
086 CF7E B7             MPT51  ORA   A         Nulstring ?
087 CF7F CA7CEB                 JZ    :EB7C    (0) Then 0 into MACC
088 CF82 23                     INX   H        Else:
089 CF83 C3C7EA                 JMP   :EAC7    (0) Next byte from MEM
090                                            into MACC
091                     *
092                     ********************************************
093                     * TABLE PREFIXES FOR UNITARY OPERATIONS *
094                     ********************************************
095                     *
096                     * Only used by LIST for decoding.
097                     * The prefix gives a code for unitary operators,
098                     * which on encoding are directly determined, not
099                     * looked up in a table.
100                     *
101                     * Format: length of name / name / 5-bit opcode
102                     *
103                     OPTBB
104 CF86 01             SBRA   DATA  :01
105 CF87 28                    DATA  :28       (
106 CF88 1A                    DATA  :1A
107                     *
108 CF89 01                    DATA  :01
109 CF8A 2D                    DATA  :2D       -
110 CF8B 1D                    DATA  :1D
111                     *
112 CF8C 01                    DATA  :01
113 CF8D 2B                    DATA  :2B       +
114 CF8E 1C                    DATA  :1C
115                     *
116 CF8F 00                    DATA  :00       Fixed FPT conversion
117 CF90 1F                    DATA  :1F
118                     *
119                     ***************************
120                     * TABLE BINARY OPERATORS *
121                     ***************************
122                     *
123                     * Format: 1 byte:  length of name.
124                     *         n bytes: name.
                                  1 byte:  code byte:
```

```
                        *                       highest 3 bits: priority.
                        *                       lowest  5 bits: opcode.
                        *
                        OPTAB
CF91 01                 SDIV    DATA  :01
CF92 2F                         DATA  :2F       /
CF93 C2                         DATA  :C2
                        *
CF94 01                         DATA  :01
CF95 2A                         DATA  :2A       *
CF96 C3                         DATA  :C3
                        *
CF97 03                         DATA  :03
CF98 4D                         DATA  :4D       M
CF99 4F                         DATA  :4F       O
CF9A 44                         DATA  :44       D
CF9B CF                         DATA  :CF
                        *
CF9C 01                         DATA  :01
CF9D 5E                         DATA  :5E       ^
CF9E E4                         DATA  :E4
                        *
CF9F 04                         DATA  :04
CFA0 49                         DATA  :49       I
CFA1 41                         DATA  :41       A
CFA2 4E                         DATA  :4E       N
CFA3 44                         DATA  :44       D
CFA4 69                         DATA  :69
                        *
CFA5 03                         DATA  :03
CFA6 49                         DATA  :49       I
CFA7 4F                         DATA  :4F       O
CFA8 52                         DATA  :52       R
CFA9 6A                         DATA  :6A
                        *
CFAA 04                         DATA  :04
CFAB 49                         DATA  :49       I
CFAC 58                         DATA  :58       X
CFAD 4F                         DATA  :4F       O
CFAE 52                         DATA  :52       R
CFAF 6C                         DATA  :6C
                        *
CFB0 03                         DATA  :03
CFB1 53                         DATA  :53       S
CFB2 48                         DATA  :48       H
CFB3 4C                         DATA  :4C       L
CFB4 8D                         DATA  :8D
                        *
CFB5 03                         DATA  :03
CFB6 53                         DATA  :53       S
CFB7 48                         DATA  :48       H
CFB8 52                         DATA  :52       R
CFB9 8E                         DATA  :8E
                        *
CFBA 02                         DATA  :02
CFBB 3E                         DATA  :3E       >
CFBC 3D                         DATA  :3D       =
CFBD 50                         DATA  :50
                        *
CFBE 01                         DATA  :01
CFBF 3E                         DATA  :3E       >
CFC0 51                         DATA  :51
```

```
188                     *
189 CFC1 02             *       DATA    :02
190 CFC2 3C             *       DATA    :3C     <
191 CFC3 3E             *       DATA    :3E     >
192 CFC4 52             *       DATA    :52
193                     *
194 CFC5 02             *       DATA    :02
195 CFC6 3C             *       DATA    :3C     <
196 CFC7 3D             *       DATA    :3D     =
197 CFC8 53             *       DATA    :53
198                     *
199 CFC9 01             *       DATA    :01
200 CFCA 3C             *       DATA    :3C     <
201 CFCB 54             *       DATA    :54
202                     *
203 CFCC 01             *       DATA    :01
204 CFCD 3D             *       DATA    :3D     >
205 CFCE 55             *       DATA    :55
206                     *
207 CFCF 03             *SAND   DATA    :03
208 CFD0 41             *       DATA    :41     A
209 CFD1 4E             *       DATA    :4E     N
210 CFD2 44             *       DATA    :44     D
211 CFD3 38             *       DATA    :38
212                     *
213 CFD4 02             *       DATA    :02
214 CFD5 4F             *       DATA    :4F     O
215 CFD6 52             *       DATA    :52     R
216 CFD7 39             *       DATA    :39
217                     *
218                     ******************************
219                     * TABLE UNITARY OPERATORS *
220                     ******************************
221                     *
222                     * Format: 1 byte:  Length of name.
223                     *         n bytes: Name.
224                     *         1 byte:  Code byte:
225                     *                  3 highest bits: priority.
226                     *                  5 lowest bits:  opcode.
227                     *
228 CFD8 04             *OPTBM  DATA    :04
229 CFD9 49             *       DATA    :49     I
230 CFDA 4E             *       DATA    :4E     N
231 CFDB 4F             *       DATA    :4F     O
232 CFDC 54             *       DATA    :54     T
233 CFDD 1E             *       DATA    :1E
234                     *
235 CFDE 01             *       DATA    :01
236 CFDF 2B             *       DATA    :2B     +
237 CFE0 A0             *       DATA    :A0
238                     *
239 CFE1 01             *       DATA    :01
240 CFE2 2D             *       DATA    :2D     -
241 CFE3 A1             *       DATA    :A1
242                     *
243 CFE4 00             *       DATA    :00     End of table
244 CFE5 00             *       DATA    :00
245                     *
246                     ***********************************
247                     * TABLE STRINGS BASIC FUNCTIONS *
248                     ***********************************
249                     *
```

```
                        * Format: 1 byte:  Length of name.
                        *         n bytes: Name.
                        *         1 byte:  High nibble: type of info.
                        *                  Low nibble: nr of arguments.
                        *         1 byte:  High nibble: required type
                        *                  of variable expected:
                        *                  (0=FPT, 1=INT, 2=STR).
                        *
                        FUNTB
CFE6 03                 SABS    DATA  :03
CFE7 41                         DATA  :41       A
CFE8 42                         DATA  :42       B
CFE9 53                         DATA  :53       S
CFEA 01                         DATA  :01
CFEB 00                         DATA  :00
                        *
CFEC 04                 SALOG   DATA  :04
CFED 41                         DATA  :41       A
CFEE 4C                         DATA  :4C       L
CFEF 4F                         DATA  :4F       O
CFF0 47                         DATA  :47       G
CFF1 01                         DATA  :01
CFF2 00                         DATA  :00
                        *
CFF3 03                 SASC    DATA  :03
CFF4 41                         DATA  :41       A
CFF5 53                         DATA  :53       S
CFF6 43                         DATA  :43       C
CFF7 11                         DATA  :11
CFF8 20                         DATA  :20
                        *
CFF9 04                 SCHR    DATA  :04
CFFA 43                         DATA  :43       C
CFFB 48                         DATA  :48       H
CFFC 52                         DATA  :52       R
CFFD 24                         DATA  :24       $
CFFE 21                         DATA  :21
CFFF 10                         DATA  :10
                        *
D000 04                 SCURX   DATA  :04
D001 43                         DATA  :43       C
D002 55                         DATA  :55       U
D003 52                         DATA  :52       R
D004 58                         DATA  :58       X
D005 10                         DATA  :10
                        *
D006 04                 SCURY   DATA  :04
D007 43                         DATA  :43       C
D008 55                         DATA  :55       U
D009 52                         DATA  :52       R
D00A 59                         DATA  :59       Y
D00B 10                         DATA  :10
                        *
D00C 03                 SEXP    DATA  :03
D00D 45                         DATA  :45       E
D00E 58                         DATA  :58       X
D00F 50                         DATA  :50       P
D010 01                         DATA  :01
D011 00                         DATA  :00
                        *
D012 04                 SFRAC   DATA  :04
D013 46                         DATA  :46       F
```

```
312 D014 52                     DATA  :52       R
313 D015 41                     DATA  :41       A
314 D016 43                     DATA  :43       C
315 D017 01                     DATA  :01
316 D018 00                     DATA  :00
317                     *
318 D019 03             SFRE    DATA  :03
319 D01A 46                     DATA  :46       F
320 D01B 52                     DATA  :52       R
321 D01C 45                     DATA  :45       E
322 D01D 10                     DATA  :10
323                     *
324 D01E 04             SFREQ   DATA  :04
325 D01F 46                     DATA  :46       F
326 D020 52                     DATA  :52       R
327 D021 45                     DATA  :45       E
328 D022 51                     DATA  :51       Q
329 D023 11                     DATA  :11
330 D024 00                     DATA  :00
331                     *
332 D025 04             SGETC   DATA  :04
333 D026 47                     DATA  :47       G
334 D027 45                     DATA  :45       E
335 D028 54                     DATA  :54       T
336 D029 43                     DATA  :43       C
337 D02A 10                     DATA  :10
338                     *
339 D02B 04             SHEX    DATA  :04
340 D02C 48                     DATA  :48       H
341 D02D 45                     DATA  :45       E
342 D02E 58                     DATA  :58       X
343 D02F 24                     DATA  :24       $
344 D030 21                     DATA  :21
345 D031 10                     DATA  :10
346                     *
347 D032 03             SINP    DATA  :03
348 D033 49                     DATA  :49       I
349 D034 4E                     DATA  :4E       N
350 D035 50                     DATA  :50       P
351 D036 11                     DATA  :11
352 D037 10                     DATA  :10
353                     *
354 D038 03             SINT    DATA  :03
355 D039 49                     DATA  :49       I
356 D03A 4E                     DATA  :4E       N
357 D03B 54                     DATA  :54       T
358 D03C 01                     DATA  :01
359 D03D 00                     DATA  :00
360                     *
361 D03E 05             SLEFT   DATA  :05
362 D03F 4C                     DATA  :4C       L
363 D040 45                     DATA  :45       E
364 D041 46                     DATA  :46       F
365 D042 54                     DATA  :54       T
366 D043 24                     DATA  :24       $
367 D044 22                     DATA  :22
368 D045 20                     DATA  :20
369 D046 10                     DATA  :10
370                     *
371 D047 03             SLEN    DATA  :03
372 D048 4C                     DATA  :4C       L
373 D049 45                     DATA  :45       E
```

```
374 D04A 4E                     DATA  :4E       N
375 D04B 11                     DATA  :11
376 D04C 20                     DATA  :20
377                     *
378 D04D 06             SVPT    DATA  :06
379 D04E 56                     DATA  :56       V
380 D04F 41                     DATA  :41       A
381 D050 52                     DATA  :52       R
382 D051 50                     DATA  :50       P
383 D052 54                     DATA  :54       T
384 D053 52                     DATA  :52       R
385 D054 11                     DATA  :11
386 D055 30                     DATA  :30
387                     *
388 D056 03             SLOG    DATA  :03
389 D057 4C                     DATA  :4C       L
390 D058 4F                     DATA  :4F       O
391 D059 47                     DATA  :47       G
392 D05A 01                     DATA  :01
393 D05B 00                     DATA  :00
394                     *
395 D05C 04             SLOGT   DATA  :04
396 D05D 4C                     DATA  :4C       L
397 D05E 4F                     DATA  :4F       O
398 D05F 47                     DATA  :47       G
399 D060 54                     DATA  :54       T
400 D061 01                     DATA  :01
401 D062 00                     DATA  :00
402                     *
403 D063 04             SXMAX   DATA  :04
404 D064 58                     DATA  :58       X
405 D065 4D                     DATA  :4D       M
406 D066 41                     DATA  :41       A
407 D067 58                     DATA  :58       X
408 D068 10                     DATA  :10
409                     *
410 D069 04             SYMAX   DATA  :04
411 D06A 59                     DATA  :59       Y
412 D06B 4D                     DATA  :4D       M
413 D06C 41                     DATA  :41       A
414 D06D 58                     DATA  :58       X
415 D06E 10                     DATA  :10
416                     *
417 D06F 04             SMID    DATA  :04
418 D070 4D                     DATA  :4D       M
419 D071 49                     DATA  :49       I
420 D072 44                     DATA  :44       D
421 D073 24                     DATA  :24       $
422 D074 23                     DATA  :23
423 D075 20                     DATA  :20
424 D076 10                     DATA  :10
425 D077 10                     DATA  :10
426                     *
427 D078 03             SPDL    DATA  :03
428 D079 50                     DATA  :50       P
429 D07A 44                     DATA  :44       D
430 D07B 4C                     DATA  :4C       L
431 D07C 11                     DATA  :11
432 D07D 10                     DATA  :10
433                     *
434 D07E 04             SPEEK   DATA  :04
435 D07F 50                     DATA  :50       P
```

```
436 D080 45                     DATA  :45       E
437 D081 45                     DATA  :45       E
438 D082 4B                     DATA  :4B       K
439 D083 11                     DATA  :11
440 D084 10                     DATA  :10
441                     *
442 D085 02             SPI     DATA  :02
443 D086 50                     DATA  :50       P
444 D087 49                     DATA  :49       I
445 D088 00                     DATA  :00
446                     *
447 D089 06             SRIGHT  DATA  :06
448 D08A 52                     DATA  :52       R
449 D08B 49                     DATA  :49       I
450 D08C 47                     DATA  :47       G
451 D08D 48                     DATA  :48       H
452 D08E 54                     DATA  :54       T
453 D08F 24                     DATA  :24       $
454 D090 22                     DATA  :22
455 D091 20                     DATA  :20
456 D092 10                     DATA  :10
457                     *
458 D093 03             SRND    DATA  :03
459 D094 52                     DATA  :52       R
460 D095 4E                     DATA  :4E       N
461 D096 44                     DATA  :44       D
462 D097 01                     DATA  :01
463 D098 00                     DATA  :00
464                     *
465 D099 04             SSCRN   DATA  :04
466 D09A 53                     DATA  :53       S
467 D09B 43                     DATA  :43       C
468 D09C 52                     DATA  :52       R
469 D09D 4E                     DATA  :4E       N
470 D09E 12                     DATA  :12
471 D09F 10                     DATA  :10
472 D0A0 10                     DATA  :10
473                     *
474 D0A1 03             SSGN    DATA  :03
475 D0A2 53                     DATA  :53       S
476 D0A3 47                     DATA  :47       G
477 D0A4 4E                     DATA  :4E       N
478 D0A5 01                     DATA  :01
479 D0A6 00                     DATA  :00
480                     *
481 D0A7 03             SSPC    DATA  :03
482 D0A8 53                     DATA  :53       S
483 D0A9 50                     DATA  :50       P
484 D0AA 43                     DATA  :43       C
485 D0AB 21                     DATA  :21
486 D0AC 10                     DATA  :10
487                     *
488 D0AD 03             SSQR    DATA  :03
489 D0AE 53                     DATA  :53       S
490 D0AF 51                     DATA  :51       Q
491 D0B0 52                     DATA  :52       R
492 D0B1 01                     DATA  :01
493 D0B2 00                     DATA  :00
494                     *
495 D0B3 04             SSTR    DATA  :04
496 D0B4 53                     DATA  :53       S
497 D0B5 54                     DATA  :54       T
```

```
498 D0B6 52                 DATA  :52       R
499 D0B7 24                 DATA  :24       $
500 D0B8 21                 DATA  :21
501 D0B9 00                 DATA  :00
502                 *
503 D0BA 03         STAB    DATA  :03
504 D0BB 54                 DATA  :54       T
505 D0BC 41                 DATA  :41       A
506 D0BD 42                 DATA  :42       B
507 D0BE 21                 DATA  :21
508 D0BF 10                 DATA  :10
509                 *
510 D0C0 03         SVAL    DATA  :03
511 D0C1 56                 DATA  :56       V
512 D0C2 41                 DATA  :41       A
513 D0C3 4C                 DATA  :4C       L
514 D0C4 01                 DATA  :01
515 D0C5 20                 DATA  :20
516                 *
517 D0C6 03         SSIN    DATA  :03
518 D0C7 53                 DATA  :53       S
519 D0C8 49                 DATA  :49       I
520 D0C9 4E                 DATA  :4E       N
521 D0CA 01                 DATA  :01
522 D0CB 00                 DATA  :00
523                 *
524 D0CC 03         SCOS    DATA  :03
525 D0CD 43                 DATA  :43       C
526 D0CE 4F                 DATA  :4F       O
527 D0CF 53                 DATA  :53       S
528 D0D0 01                 DATA  :01
529 D0D1 00                 DATA  :00
530                 *
531 D0D2 03         STAN    DATA  :03
532 D0D3 54                 DATA  :54       T
533 D0D4 41                 DATA  :41       A
534 D0D5 4E                 DATA  :4E       N
535 D0D6 01                 DATA  :01
536 D0D7 00                 DATA  :00
537                 *
538 D0D8 04         SASIN   DATA  :04
539 D0D9 41                 DATA  :41       A
540 D0DA 53                 DATA  :53       S
541 D0DB 49                 DATA  :49       I
542 D0DC 4E                 DATA  :4E       N
543 D0DD 01                 DATA  :01
544 D0DE 00                 DATA  :00
545                 *
546 D0DF 04         SACOS   DATA  :04
547 D0E0 41                 DATA  :41       A
548 D0E1 43                 DATA  :43       C
549 D0E2 4F                 DATA  :4F       O
550 D0E3 53                 DATA  :53       S
551 D0E4 01                 DATA  :01
552 D0E5 00                 DATA  :00
553                 *
554 D0E6 03         SATN    DATA  :03
555 D0E7 41                 DATA  :41       A
556 D0E8 54                 DATA  :54       T
557 D0E9 4E                 DATA  :4E       N
558 D0EA 01                 DATA  :01
559 D0EB 00                 DATA  :00
```

```
                        *
D0EC 00                 DATA  :00       End of table
                        *
                        ********
                        * DATA *
                        ********
                        *
                        ENDFT
D0ED 15                 FPOSC DATA  :15       Sound constant
D0EE F4                       DATA  :F4
D0EF 24                       DATA  :24
D0F0 00                       DATA  :00
                        *
D0F1 81                 FPM1  DATA  :81       FPT (-1)
D0F2 80                       DATA  :80
D0F3 00                       DATA  :00
D0F4 00                       DATA  :00
                        *
D0F5 02                 FPPI  DATA  :02       FPT (PI)
D0F6 C9                       DATA  :C9
D0F7 0F                       DATA  :0F
D0F8 DB                       DATA  :DB
                        *
D0F9 00                 I4    DATA  :00       INT (4)  (not used)
D0FA 00                       DATA  :00
D0FB 00                       DATA  :00
D0FC 04                       DATA  :04
                        *
D0FD 00                 IRAND DATA  :00       AND mask
D0FE FF                       DATA  :FF
D0FF FF                       DATA  :FF
D100 FF                       DATA  :FF
                        *
                        *
                        *
D101                          END
```

```
****************************
* S Y M B O L   T A B L E *
****************************
```

CITAB	CF02	ENDFT	D0ED	FPM1	D0F1	FPOSC	D0ED
FPPI	D0F5	FUNTB	CFE6	I4	D0F9	IRAND	D0FD
MPT51	CF7E	OPTAB	CF91	OPTBB	CF86	OPTBM	CFD8
SABS	CFE6	SACOS	D0DF	SALOG	CFEC	SAND	CFCF
SASC	CFF3	SASIN	D0D8	SATN	D0E6	SBRA	CF86
SCHR	CFF9	SCOS	D0CC	SCURX	D000	SCURY	D006
SDIV	CF91	SEXP	D00C	SFRAC	D012	SFRE	D019
SFREQ	D01E	SGETC	D025	SHEX	D02B	SINP	D032
SINT	D038	SLEFT	D03E	SLEN	D047	SLOG	D056
SLOGT	D05C	SMID	D06F	SPDL	D078	SPEEK	D07E
SPI	D085	SRIGHT	D089	SRND	D093	SSCRN	D099
SSGN	D0A1	SSIN	D0C6	SSPC	D0A7	SSQR	D0AD
SSTR	D0B3	STAB	D0BA	STAN	D0D2	SVAL	D0C0
SVPT	D04D	SXMAX	D063	SYMAX	D069		

```
002                             ORG   :D101
003                     *
004                     *
005                     *
006                     *  ===============
007                     *** STRING HANDLER ***
008                     *  ===============
009                     *
010                     *
011                     ******************************
012                     * INITIALISE STRING HANDLER *
013                     ******************************
014                     *
015                     * Initialises a given size of string area.
016                     * This routine is used once by RESET (C719), but
017                     * without purpose. It belongs to another BASIC
018                     * package of the firm DAI.
019                     *
020                     * Entry: None.
021                     * Exit:  A=0, BCDEHL preserved. F corrupted.
022                     *
023 D101 AF             SHINIT  XRA   A
024 D102 320000                 STA   :0000     Zero 0000
025 D105 C9                     RET
026                     *
027                     ***********************
028                     * APPEND TWO STRINGS *
029                     ***********************
030                     *
031                     * Appends two strings together to one new string
032                     * by adding the 2nd string to the end of the 1st
033                     * string.
034                     *
035                     * Entry: DE: Startaddress 1st string.
036                     *            (1st byte is length byte).
037                     *        HL: Startaddress 2nd string.
038                     * Exit:  AFBCDE preserved.
039                     *        HL: Points to new string.
040                     *        Error if string too long.
041                     *
042 D106 F5             SHAPP   PUSH  PSW
043 D107 D5                     PUSH  D
044 D108 1A                     LDAX  D         Get length 1st string
045 D109 86                     ADD   M         Calc length new string
046 D10A DA38DA                 JC    :DA38     Run error 'STRING TOO LONG'
047                                             if length >255.
048 D10D E5                     PUSH  H         Save pntr 2nd string
049 D10E CD8BD1                 CALL  :D18B     Find place in Heap for
050                                             new string
051 D111 E5                     PUSH  H         Save pntr to new string
052 D112 23                     INX   H
053 D113 CD6DD1                 CALL  :D16D     Move 1st string to new loc.
054 D116 D1                     POP   D
055 D117 E3                     XTHL
056 D118 EB                     XCHG
057 D119 E3                     XTHL
058 D11A CD6DD1                 CALL  :D16D     Move 2nd string to new loc.
059 D11D E1                     POP   H         Get pntr to new string
060 D11E D1                     POP   D
061 D11F F1                     POP   PSW
062 D120 C9                     RET
063                     *
```

```
064                     ************************
065                     * COMPARE TWO STRINGS *
066                     ************************
067                     *
068                     * Compares two string character by character.
069                     * No characters >=#80 are allowed.
070                     *
071                     * Entry: DE: beginaddr. 1st string.
072                     *        HL: beginaddr. 2nd string.
073                     * Exit:  ABCDEHL preserved.
074                     *        Flags:     Z=1: Both strings ident.
075                     *                 S=1,Z=0: 2nd string longer.
076                     *                 S=0,Z=0: 1st string longer.
077                     *
078 D121 C5              SHCOMP  PUSH  B
079 D122 F5                      PUSH  PSW
080 D123 D5                      PUSH  D
081 D124 E5                      PUSH  H
082 D125 1A                      LDAX  D          Get length 1st string
083 D126 47                      MOV   B,A        Store it in B
084 D127 4E                      MOV   C,M        Length 2nd string in C
085 D128 13              LD3     INX   D
086 D129 23                      INX   H
087 D12A 78                      MOV   A,B
088 D12B D601                    SUI   :01        Check 1st string empty
089 D12D 47                      MOV   B,A        Save rest of bytes
090 D12E DA45D1                  JC    :D145      If 1st string empty
091 D131 79                      MOV   A,C
092 D132 D601                    SUI   :01        Check 2nd string empty
093 D134 4F                      MOV   C,A        Save rest of bytes
094 D135 3C                      INR   A
095 D136 3C                      INR   A
096 D137 DA3FD1                  JC    :D13F      If 2nd string empty
097 D13A 1A                      LDAX  D          Get byte 1st string
098 D13B BE                      CMP   M          and compare it with 2nd
099 D13C CA28D1                  JZ    :D128      If identical: cont. test
100 D13F E1              LD4     POP   H
101 D140 D1                      POP   D
102 D141 C1                      POP   B
103 D142 78                      MOV   A,B        Restore A
104 D143 C1                      POP   B
105 D144 C9                      RET
106
107                     * If 1st string empty:
108
109 D145 79              LD5     MOV   A,C        Get length 2nd string
110 D146 B7                      ORA   A
111 D147 CA3FD1                  JZ    :D13F      If also empty: abort, Z=1
112 D14A AF                      XRA   A
113 D14B 3D                      DCR   A          Z=0
114 D14C C33FD1                  JMP   :D13F      Quit
115                     *
116                     ******************************************
117                     * EXTRACT A SUBSTRING FROM THE MIDDLE OF *
118                     * ANOTHER STRING                         *
119                     ******************************************
120                     *
121                     * Entry: HL points to string.
122                     *        D  offset substring.
123                     *        E  length substring.
124                     * Exit:  If O.K.: HL points to substring.
125                     *        Else: Jump to error.
```

```
126                     *       AF corrupted. BCDE preserved.
127                     *
128 D14F C5             SHMID   PUSH  B
129 D150 D5                     PUSH  D
130 D151 7E             SHM05   MOV   A,M       Get length string
131 D152 92                     SUB   D         Minus offset substring
132 D153 DA15DA                 JC    :DA15     Run error 'NUMBER OUT OF
133                                               RANGE' if offset too big
134 D156 93                     SUB   E         Minus length substring
135 D157 DA15DA                 JC    :DA15     Run error 'NUMBER OUT OF
136                                               RANGE' if length too big
137 D15A 7A             SHM08   MOV   A,D       Get offset
138 D15B CD30DE                 CALL  :DE30     Calc beginaddr substring
139 D15E 23                     INX   H
140 D15F E5                     PUSH  H         Save begin substring
141 D160 7B                     MOV   A,E       Get length substring
142 D161 CD8BD1                 CALL  :D18B     Find place in Heap for
143                                               substring
144 D164 D1                     POP   D         Get begin substring
145 D165 E5                     PUSH  H
146 D166 CD73D1                 CALL  :D173     Move substring into Heap
147 D169 E1                     POP   H
148 D16A D1                     POP   D
149 D16B C1                     POP   B
150 D16C C9                     RET
151                     *
152                     ****************************
153                     * TRANSFER OF STRING DATA *
154                     ****************************
155                     *
156                     * Copies strings from one place to another.
157                     *
158                     * Start at SCOPF: Transfer known string from DE
159                     *                 to HL.
160                     * Start at SCOPT: Transfer string into limited
161                     *                 space of HL.
162                     *
163                     * Entry: DE: Startaddr. string to be transferred.
164                     *        HL: Destination address.
165                     *            1st bytes are length bytes.
166                     * Exit:  Both DE + HL point to byte after string.
167                     *        BC preserved; A=FF, CY=1.
168                     *
169 D16D 1A             SCOPF   LDAX  D         Get length
170 D16E 13                     INX   D         Pnt to 1st byte
171 D16F C375D1                 JMP   :D175
172                     *
173 D172 13             SHCOPY  INX   D         Pnt to 1st byte of string
174 D173 7E             SCOPT   MOV   A,M       Get available space
175 D174 23                     INX   H         Pnt to 1st place to store
176 D175 C5             LD9     PUSH  B
177 D176 47                     MOV   B,A       Available space/space reqd
178                                               in B
179 D177 78             LD10    MOV   A,B       Get space still available
180 D178 D601                   SUI   :01       Decr. space available
181 D17A 47                     MOV   B,A       and save it
182 D17B DA85D1                 JC    :D185     Jump if ready
183 D17E 1A                     LDAX  D         Get byte to be transferred
184 D17F 77                     MOV   M,A       and transfer it
185 D180 13                     INX   D         Pnt to next byte
186 D181 23                     INX   H         Pnt to next place
187 D182 C377D1                 JMP   :D177     Transfer next byte
```

```
188                     *
189 D185 C1             LD11    POP   B
190 D186 C9                     RET
191                     *
192                     ***************************
193                     * RELINGUISH HEAP SPACE *
194                     ***************************
195                     *
196                     * Clears a string in the heap.
197                     *
198                     * Entry: HL points to 2nd length byte of string.
199                     * Exit:  HL points to 2nd length byte of cleared
200                     *        string. AFBCDE preserved.
201                     *
202 D187 2B             SHREL   DCX   H
203 D188 C336D2                 JMP   :D236     Clear Heap entry.
204                     *
205                     ****************************************
206                     * REQUEST SPACE IN HEAP FOR A STRING *
207                     ****************************************
208                     *
209                     * Finds a place in the heap for a new string.
210                     *
211                     * Entry: A: Required space.
212                     * Exit:  AFBCDE preserved.
213                     *        HL points to length of reserved area.
214                     *        Error if no space available.
215                     *
216 D18B D5             SHREQ   PUSH  D
217 D18C 1600                   MVI   D,:00     ) Required space in DE
218 D18E 5F                     MOV   E,A       )
219 D18F CDC5D1                 CALL  :D1C5     Run Heap request
220 D192 23                     INX   H
221 D193 D1                     POP   D
222 D194 C9                     RET
223                     *
224                     *
225                     *  ==============
226                     *** HEAP HANDLER ***
227                     *  ==============
228                     *
229                     *
230                     ***************************************
231                     * INIT. HEAP SPACE TO ALL AVAILABLE *
232                     ***************************************
233                     *
234                     * The pointers for textbuffer and symboltable
235                     * are updated correctly according to the requested
236                     * Heapsize. The Heap is cleared: It starts with
237                     * HSIZE-4 IOR #8000 and ends with #7FFF.
238                     *
239                     * Entry: DE: HEAP size (<32K).
240                     * Exit:  BC preserved. AFDEHL corrupted.
241                     *        Error if insufficient space.
242                     *
243 D195 2A9F02        HINIT   LHLD  :029F     Start text buffer in HL
244 D198 D5                   PUSH  D
245 D199 E5                   PUSH  H
246 D19A D5                   PUSH  D
247 D19B EB                   XCHG            Start textbuf in DE
248 D19C 2A9B02               LHLD  :029B     Start Heap in HL
        EB                    XCHG
```

```
188                     *
189 D185 C1             LD11    POP   B
190 D186 C9                     RET
191                     *
192                     ***************************
193                     * RELINGUISH HEAP SPACE *
194                     ***************************
195                     *
196                     * Clears a string in the heap.
197                     *
198                     * Entry: HL points to 2nd length byte of string.
199                     * Exit:  HL points to 2nd length byte of cleared
200                     *        string. AFBCDE preserved.
201                     *
202 D187 2B             SHREL   DCX   H
203 D188 C336D2                 JMP   :D236     Clear Heap entry.
204                     *
205                     ****************************************
206                     * REQUEST SPACE IN HEAP FOR A STRING *
207                     ****************************************
208                     *
209                     * Finds a place in the heap for a new string.
210                     *
211                     * Entry: A: Required space.
212                     * Exit:  AFBCDE preserved.
213                     *        HL points to length of reserved area.
214                     *        Error if no space available.
215                     *
216 D18B D5             SHREQ   PUSH  D
217 D18C 1600                   MVI   D,:00     ) Required space in DE
218 D18E 5F                     MOV   E,A       )
219 D18F CDC5D1                 CALL  :D1C5     Run Heap request
220 D192 23                     INX   H
221 D193 D1                     POP   D
222 D194 C9                     RET
223                     *
224                     *
225                     *  ==============
226                     *** HEAP HANDLER ***
227                     *  ==============
228                     *
229                     *
230                     ***************************************
231                     * INIT. HEAP SPACE TO ALL AVAILABLE *
232                     ***************************************
233                     *
234                     * The pointers for textbuffer and symboltable
235                     * are updated correctly according to the requested
236                     * Heapsize. The Heap is cleared: It starts with
237                     * HSIZE-4 IOR #8000 and ends with #7FFF.
238                     *
239                     * Entry: DE: HEAP size (<32K).
240                     * Exit:  BC preserved. AFDEHL corrupted.
241                     *        Error if insufficient space.
242                     *
243 D195 2A9F02        HINIT   LHLD  :029F     Start text buffer in HL
244 D198 D5                   PUSH  D
245 D199 E5                   PUSH  H
246 D19A D5                   PUSH  D
247 D19B EB                   XCHG            Start textbuf in DE
248 D19C 2A9B02               LHLD  :029B     Start Heap in HL
        EB                    XCHG
```

```
250 D1A0 CD1ADE             CALL  :DE1A     Calc current heapsize
251 D1A3 EB                 XCHG            Store it in DE
252 D1A4 E1                 POP   H         Get new HEAP size
253 D1A5 CD1ADE             CALL  :DE1A     Calculate difference
254 D1A8 D1                 POP   D         Get old start textbuf
255 D1A9 CDD1C9             CALL  :C9D1     Move textbuf and symtab
256 D1AC 229F02             SHLD  :029F     Update start textbuf
257 D1AF D1                 POP   D         Get new HEAP size
258 D1B0 2A9B02             LHLD  :029B     Get startaddr HEAP
259 D1B3 1B                 DCX   D
260 D1B4 1B                 DCX   D
261 D1B5 1B                 DCX   D
262 D1B6 1B                 DCX   D         HSIZE-4
263 D1B7 7A                 MOV   A,D
264 D1B8 F680               ORI   :80       Free space available
265 D1BA 77                 MOV   M,A       ) Set start Heap to
266 D1BB 23                 INX   H         ) HSIZE-4 IOR #8000
267 D1BC 73                 MOV   M,E       )
268 D1BD 23                 INX   H
269 D1BE 19                 DAD   D         Get end HEAP -2
270 D1BF 367F               MVI   M,:7F     ) Set it to
271 D1C1 23                 INX   H         ) #7FFF
272 D1C2 36FF               MVI   M,:FF     ) (end marker)
273 D1C4 C9                 RET
274                 *
275                 ****************
276                 * HEAP REQUEST *
277                 ****************
278                 *
279 D1C5 CF         HREQU   RST   1         Run heap request
280 D1C6 0F                 DATA  :0F
281 D1C7 C9                 RET
282                 *
283 D1C8 FF                 DATA  :FF
284 D1C9 FF                 DATA  :FF
285 D1CA FF                 DATA  :FF
286 D1CB FF                 DATA  :FF
287 D1CC FF                 DATA  :FF
288 D1CD FF                 DATA  :FF
289 D1CE FF                 DATA  :FF
290 D1CF FF                 DATA  :FF
291 D1D0 FF                 DATA  :FF
292                 *
293                 ************************
294                 * part of EDIT (2EEC0) *
295                 ************************
296                 *
297                 * Prints a text line.
298                 *
299 D1D1 CAECEE     LD15    JZ    :EEEC     (2) If char=0
300 D1D4 F2DEEE             JP    :EEDE     (2) Cont for char #01-7F
301 D1D7 23                 INX   H         Next pos in textbuf
302 D1D8 C3D2EE             JMP   :EED2     (2) Ignore char >= #80
303                 *
304                 ******************************************
305                 * INPUT SCANNING: CHECK SOURCE AND INPUTS *
306                 ******************************************
307                 *
308                 * Part of D6BB.
309                 * Checks if input from keyboard or from DINC.
310                 * Checks also for any new inputs.
311                 *
```

```
312 D1DB 3A9602      MPT29   LDA   :0296     Get input direction
313 D1DE B7                  ORA   A
314 D1DF C2E002              JNZ   :02E0     Jump if input from DINC
315
316                  * If input from keyboard:
317
318 D1E2 CDA5D6      IN0     CALL  :D6A5     Check if new keys pressed
319 D1E5 C3C1D6              JMP   :D6C1     Into keyb.scanning
320                  *
321                  ********************************
322                  * part of FPT COMPARE (C079) *
323                  ********************************
324                  *
325                  * Entry: Contents MACC in ABCD, exp.byte in E.
326                  *         HL points to exp.byte MEM.
327                  *
328 D1E8 78           LD17    MOV   A,B
329 D1E9 23                   INX   H
330 D1EA A6                   ANA   M         AND 1st bytes both mantissas
331 D1EB 7E                   MOV   A,M
332 D1EC 2B                   DCX   H         Fnts to exp.byte MEM
333 D1ED FAF5D1               JM    :D1F5     Jump if both 1st mantissa
334                                            bits are '1'
335 D1F0 B8                   CMP   B         Comp both 1th mantissa bytes
336 D1F1 3F                   CMC             CY=0 if mantissa MACC >
337                                            mantissa MEM
338 D1F2 C39FC0               JMP   :C09F     Quit
339                  *
340 D1F5 7B           LD18    MOV   A,E       Get exp.byte MACC
341 D1F6 AE                   XRA   M         XOR both exponents
342 D1F7 E640                 ANI   :40       Check if only 1 exp. neg.
343 D1F9 7B                   MOV   A,E       Get exp.byte MACC
344 D1FA C387C0               JMP   :C087     Back to main routine
345                  *
346                  ************************
347                  * part of EDIT (2EE7B) *
348                  ************************
349                  *
350                  * Skip to B'th position on line.
351                  *
352 D1FD 7E           LD19    MOV   A,M       Get char in A
353 D1FE B7                   ORA   A         Set flags on it
354 D1FF FA94EE               JM    :EE94     (2) Ignore char >= #80
355 D202 78                   MOV   A,B       Get pos. required
356 D203 B9                   CMP   C         Reached ?
357 D204 7E                   MOV   A,M       Get char
358 D205 C382EE               JMP   :EE82     (2)
359                  *
360                  ********
361                  * DATA *
362                  ********
363                  *
364 D208 0D           MSG01   DATA  :0D       CR
365 D209 DB6F                 DBL   :6FDB     'SOME INPUT IGNORED'
366 D20B 00                   DATA  :00
367                  *
368                  ************************************
369                  * part of UNDERFLOW ERROR (C065) *
370                  ************************************
371                  *
372 D20C E7           LD21    RST   4         Copy operand to MACC
373 D20D 0C                   DATA  :0C
```

```
374 D20E 210400            LXI   H,:0004
375 D211 C34FC0            JMP   :C04F      Cont on (00D0/1)+4
376                  *
377                  ****************************************
378                  * part of RUN 'CLEAR' (cont. of 0E6B5) *
379                  ****************************************
380                  *
381 D214 229D02      MPT44 SHLD  :029D      Update HEAP size
382 D217 2A0001            LHLD  :0100      Get start current line
383 D21A 7C                MOV   A,H
384 D21B B5                ORA   L          Check if CURRNT=0
385 D21C D1                POP   D
386 D21D C8                RZ               Abort if immediate cmd
387 D21E CD01E4            CALL  :E401      (0) Run RESTORE
388 D221 09                DAD   B
389 D222 CD2DE9            CALL  :E92D      (0) Calc length of block
390                                         Result in BC
391 D225 B7                ORA   A
392 D226 C9                RET
393                  *
394                  ********************************
395                  * part of HEAP REQUEST (3E9AD) *
396                  ********************************
397                  *
398                  * Checks if end of Heap reached.
399                  *
400 D227 FE7F        HRQ80 CPI   :7F        End marker ?
401 D229 C2FAE9            JNZ   :E9FA      (3) Jump if not
402 D22C 7B                MOV   A,E        Get 2nd byte in A
403 D22D 3C                INR   A
404 D22E C2FAE9            JNZ   :E9FA      (3) Jump if it was <> FF
405 D231 3E07              MVI   A,:07      If end of Heap reached:
406 D233 C3F5D9            JMP   :D9F5      Run error 'OUT OF STRING
407                                         SPACE'
408                  *
409                  ***********************
410                  * CLEAR A HEAP ENTRY *
411                  ***********************
412                  *
413                  * A Heap entry is erased by setting the msb o
414                  * the first byte to 1.
415                  *
416                  * Entry: HL: Points to byte.
417                  * Exit:  All registers preserved.
418                  *
419 D236 F5          HREL  PUSH  PSW
420 D237 7E                MOV   A,M        Get byte
421 D238 F680              ORI   :80        Set msb=1
422 D23A 77                MOV   M,A        Store byte
423 D23B F1                POP   PSW
424 D23C C9                RET
425                  *
426                  *
427                  *
428 D23D                   END
```

```
****************************
* S Y M B O L   T A B L E *
****************************

HINIT  D195   HREL   D236   HREQU  D1C5   HRQ80  D227
INO    D1E2   LD10   D177   LD11   D185   LD15   D1D1
```

LD17	D1E8	LD18	D1F5	LD19	D1FD	LD21	D20C
LD3	D128	LD4	D13F	LD5	D145	LD9	D175
MPT29	D1DB	MPT44	D214	MSG01	D208	SCOPF	D16D
SCOPT	D173	SHAPP	D106	SHCOMP	D121	SHCOPY	D172
SHINIT	D101	SHM05	D151	SHM08	D15A	SHMID	D14F
SHREL	D187	SHREQ	D18B				

```
002                             ORG   :D23D
003                     *
004                     *
005                     *
006                     *  ============
007                     *** I/O HANDLER ***
008                     *  ============
009                     *
010                     *
011                     **********************
012                     * RUN basiccmd SAVE *
013                     **********************
014                     *
015                     * Valid as direct command and in program.
016                     * Clears the Heap, zeroes all variables, evaluate
017                     * an evt. program name and writes a file of type
018                     * 0 (BASIC) on tape.
019                     *
020                     * Exit: HL: Points to end symboltable.
021                     *       DE: Length symboltable.
022                     *       BC: Updated.
023                     *
024 D23D CD23CB         RSAVE   CALL  :CB23     Empty HEAP + symtab
025 D240 2AA102                 LHLD  :02A1     Get start symtab
026 D243 EB                     XCHG            in DE
027 D244 2AA302                 LHLD  :02A3     End symtab in HL
028 D247 CD1ADE                 CALL  :DE1A     Calculate length symtab
029 D24A E5                     PUSH  H         Preserve length symtab
030 D24B 2A9F02                 LHLD  :029F     Get start textbuf
031 D24E EB                     XCHG            in DE
032 D24F CD1ADE                 CALL  :DE1A     Calculate length textbuf
033 D252 E5                     PUSH  H         Preserve length textbuf
034 D253 D5                     PUSH  D         Preserve start textbuf
035 D254 CD91E7                 CALL  :E791     (0) Evaluate program name
036 D257 00                     NOP
037 D258 00                     NOP
038 D259 00                     NOP
039 D25A 3E30                   MVI   A,:30     File type byte 0
040 D25C 00                     NOP
041 D25D 00                     NOP
042 D25E 00                     NOP
043 D25F 00                     NOP
044 D260 00                     NOP
045 D261 00                     NOP
046 D262 00                     NOP
047 D263 CDC502                 CALL  :02C5     Write fileleader, flagbyte,
048                                             file type byte, name length
049                                             and name.
050 D266 E1                     POP   H         Start textbuf in HL
051 D267 D1                     POP   D         Length textbuf in DE
052 D268 CDC802                 CALL  :02C8     Write length and contents
053                                             textbuf
054 D26B D1                     POP   D         Length symtab in DE
055 D26C C3D8D7                 JMP   :D7D8     Write length and contents
056                                             symtab + file trailer
057                     *
058 D26F FF                     DATA  :FF
059                     *
060                     **********************
061                     * RUN basiccmd LOAD *
062                     **********************
063                     *
```

```
064                     * Valid as direct command and in program.
065                     * Clears Heap and all variables. Evaluates name
066                     * of program, updates BC. Required file type: 0.
067                     * C is set to print type/name or not.
068                     * A file (type 0: Basic) is read from tape. When
069                     * a file has been found, the textbuffer and the
070                     * symboltable are loaded and the pointers updated.
071                     * When loading during program run: the program
072                     * continues with the program just loaded.
073                     *
074                     * Exit: No error: BC: Updated.
075                     *                 DE: Begin screen RAM.
076                     *                 HL: End symbol table.
077                     *
078 D270 CD23CB        RLOAD   CALL   :CB23      Empty HEAP + symtab
079 D273 CD91E7                CALL   :E791      (0) Evaluate programname
080 D276 C5                    PUSH   B
081 D277 00                    NOP
082 D278 00                    NOP
083 D279 00                    NOP
084 D27A 00                    NOP
085 D27B 0630                  MVI    B,:30      File type byte 0
086 D27D E5                    PUSH   H          Preserve length name reqd
087 D27E 2A0001                LHLD   :0100      Get CURRNT
088 D281 7C                    MOV    A,H
089 D282 B5                    ORA    L          Load during run program?
090 D283 0E00                  MVI    C,:00
091 D285 C289D2                JNZ    :D289      If during run; C=00
092 D288 0D                    DCR    C          Else C=FF
093 D289 E1            RLD10   POP    H
094 D28A CDCE02                CALL   :02CE      Switch on cassette motors;
095                                               read header + name
096 D28D 2AA502                LHLD   :02A5      Get end free RAM
097 D290 EB                    XCHG              Max. RAM in DE
098 D291 2A9F02                LHLD   :029F      Start textbuf in HL
099 D294 CDD102                CALL   :02D1      Load textbuffer
100 D297 22A102                SHLD   :02A1      Store end textbuffer
101 D29A DCD102                CC     :02D1      Load symboltable
102 D29D CD1AD7                CALL   :D71A      Store end symtab; stop
103                                               cassette motors.
104 D2A0 C1                    POP    B
105 D2A1 FB                    EI                Enable interrupts
106 D2A2 D2A8D2                JNC    :D2A8      If loading error
107 D2A5 3E00                  MVI    A,:00      No loading error
108 D2A7 C9                    RET
109                     *
110                     **********************
111                     * RUN LOADING ERROR *
112                     **********************
113                     *
114                     * The programbuffers are restored. A error message
115                     * is printed.
116                     *
117 D2A8 F5            RLERR   PUSH   PSW
118 D2A9 CDB5DE                CALL   :DEB5      Run 'NEW'
119 D2AC F1                    POP    PSW
120 D2AD 210000                LXI    H,:0000
121 D2B0 220001                SHLD   :0100      Set CURRNT=0
122 D2B3 C60B          RLEAR   ADI    :0B
123 D2B5 C3F5D9                JMP    :D9F5      Run 'LOADING ERROR ..'
124                     *
125                     *
```

```
126                     ******************
127                     * OPEN TAPE FILE *
128                     ******************
129                     *
130                     * Entry: A:  File type.
131                     *        HL: Points to file name.
132                     * Exit:  HL: Points beyond file name.
133                     *        DE: Length of name.
134                     *        BC: Preserved.
135                     *        A:  Checksum on name.
136                     *
137 D2B8 F5             CWOPEN  PUSH  PSW
138 D2B9 CD20D7                 CALL  :D720     Init. write file leader
139 D2BC F1                     POP   PSW
140 D2BD CD09D5                 CALL  :D509     Write file type byte
141 D2C0 C3F8D7                 JMP   :D7F8     Get name length, write it
142                                             on tape, incl. its c.s.
143                     *
144                     ************************
145                     * RUN basiccmd CHECK *
146                     ************************
147                     *
148                     * Valid as direct command only.
149                     * Checks on file type and name. For all files
150                     * with type <3, a checksum on all data is done.
151                     * This routine remains in a endless loop and
152                     * can be aborted with BREAK only.
153                     *
154                     RCHECK
155 D2C3 210000        CHK10   LXI   H,:0000   No program name given
156 D2C6 01FF00                 LXI   B,:00FF   Any file type
157 D2C9 00                     NOP
158 D2CA 00                     NOP
159 D2CB CDCE02                 CALL  :02CE     Read file header, file
160                                             type and name; print
161                                             type and name
162 D2CE 00                     NOP
163 D2CF FE33                   CPI   :33
164 D2D1 D2EBD2                 JNC   :D2EB     If file type >=3: no check
165                                             on checksum
166
167                     * Test checksum::
168
169 D2D4 0C                       INR   C         BC=0
170 D2D5 CDE6D7                   CALL  :D7E6     Set A=0, read + check
171                                               a data block
172 D2D8 CCD702                   CZ    :02D7     Read + check next block
173 D2DB C2E6D2                   JNZ   :D2E6     If reading error
174 D2DE CDFFDA                   CALL  :DAFF     Print 'OK', car.ret
175 D2E1 C0DB                     DBL   :DBC0
176 D2E3 C3C3D2                   JMP   :D2C3     Wait for next file
177
178                     * If checksum error:
179
180 D2E6 CDFFDA        CHK20   CALL  :DAFF     Print 'BAD'
181 D2E9 DBDB                   DBL   :DBDB
182 D2EB CD5EDD        CHK30   CALL  :DD5E     Print car.ret
183 D2EE C3C3D2                 JMP   :D2C3     Wait for next file
184                     *
185                     ************************
186                     * WRITE BLOCK ON TAPE *
187                     ************************
```

```
188                     *
189                     * Entry: HL: Startaddres block.
190                     *        DE: Length block.
191                     * Exit:  HL: 1st byte after block.
192                     *        A:  Checksum on block contents.
193                     *        BCDE preserved.
194                     *
195 D2F1 C5             CWBLK   PUSH  B
196 D2F2 D5                     PUSH  D
197 D2F3 00                     NOP
198 D2F4 CD16D3                 CALL  :D316     Write block length +
199                                             c.s. on length
200 D2F7 0656                   MVI   B,:56     Initial checksum value
201 D2F9 7A             LD34    MOV   A,D
202 D2FA B3                     ORA   E
203 D2FB CA07D3                 JZ    :D307     If all bytes written
204 D2FE 1B                     DCX   D
205 D2FF 7E                     MOV   A,M       Get byte of block
206 D300 23                     INX   H         Point to next byte
207 D301 CD0FD3                 CALL  :D30F     Write byte, update checksum
208 D304 C3F9D2                 JMP   :D2F9     Next byte
209
210                     * If all data written: write c.s. on block:
211
212 D307 78             LD35    MOV   A,B       Get calculated checksum
213 D308 CD09D5                 CALL  :D509     Write checksum
214 D30B 00                     NOP
215 D30C D1                     POP   D
216 D30D C1                     POP   B
217 D30E C9                     RET
218                     *
219                     ********************************
220                     * WRITE BYTE, UPDATE CHECKSUM *
221                     ********************************
222                     *
223                     * Entry: Byte to be written in A.
224                     *        Checksum in B.
225                     * Exit:  New checksum in B; A corrupted.
226                     *        CDEHL preserved.
227                     *
228 D30F CD09D5         LD36    CALL  :D509     Write byte
229 D312 A8                     XRA   B
230 D313 07                     RLC
231 D314 47                     MOV   B,A       Update checksum
232 D315 C9                     RET
233                     *
234                     ****************************************
235                     * WRITE BLOCK LENGTH, UPDATE CHECKSUM *
236                     ****************************************
237                     *
238                     * Entry: DE: length block.
239                     * Exit:  DEHL preserved.
240                     *
241 D316 0656           LD37    MVI   B,:56     Init checksum
242 D318 7A                     MOV   A,D       Get highest length byte,
243 D319 CD0FD3                 CALL  :D30F     write it, update c.s.
244 D31C 7B                     MOV   A,E       Get lowest length byte,
245 D31D CD0FD3                 CALL  :D30F     write it, update c.s
246 D320 78                     MOV   A,B       Get checksum
247 D321 CD09D5                 CALL  :D509     Write checksum on length
248 D324 C9                     RET
```

```
                        **********************
                        * START FILE READING *
                        **********************
                        *
                        * Entry: HL: Address length byte of name requested.
                        *         B:  File type byte requested.
                        *         C:  00 when reading during run program,
                        *             else FF.
                        * Exit:   A:  File type byte.
                        *         HL: Points to 1st byte of name requested.
                        *         DE: Length name requested.
                        *         BC: preserved.
                        *
D325 F5                 CROPEN  PUSH  PSW
D326 CDFFD7                     CALL  :D7FF     Switch cassette motors on,
                                                init. registers
D329 00                 RPN10   NOP
D32A 00                         NOP
D32B 00                         NOP
D32C 00                         NOP
D32D 00                         NOP
D32E F1                         POP   PSW
D32F CDF4D3                     CALL  :D3F4     Read file header
D332 F5                         PUSH  PSW
D333 CD8AD7                     CALL  :D78A     Display file type byte
D336 90                         SUB   B
D337 CDA2D3                     CALL  :D3A2     Read and check header,
                                                program name, file type byte
D33A B7                         ORA   A         0 if everything OK.
D33B C283D7                     JNZ   :D783     If failure
D33E F1                         POP   PSW
D33F C9                         RET
                        *
                        **************
                        * READ BLOCK *
                        **************
                        *
                        * Read length, contents and checksum of a block
                        *
                        * Entry: HL: Addr. where to dump data read.
                        *         DE: End free space.
                        * Exit:   CY=1: No error:
                        *            HL: Next free address.
                        *            BCDE preserved; AF corrupted.
                        *         CY=0: Loading error:
                        *            BCDEHL preserved.
                        *            A: Type of loading error.
                        *
D340 C5                 CRBLK   PUSH  B
D341 D5                         PUSH  D
D342 E5                         PUSH  H
D343 CD90D7                     CALL  :D790     Calculate free RAM space
D346 EB                         XCHG            Free RAM in DE
D347 CD8DD3                     CALL  :D38D     Read block length + c.s.
                                                length in HL
D34A DA7ED3                     JC    :D37E     If loading error 3
D34D B7                         ORA   A
D34E 3E00                       MVI   A,:00     Loading error 0
D350 C280D3                     JNZ   :D380     If checksum error 0
D353 E5                         PUSH  H         Save length block
D354 19                         DAD   D         Calculate free RAM
D355 D1                         POP   D         Get length block
```

```
312 D356 3C              INR   A          Loading error 1
313 D357 E1              POP   H
314 D358 E5              PUSH  H          Restore begin addr.
315 D359 DA80D3          JC    :D380      If loading error 1
316 D35C 0656            MVI   B,:56      Init checksum
317 D35E 7A        LBK10 MOV   A,D
318 D35F B3              ORA   E
319 D360 CA6FD3          JZ    :D36F      If whole block read
320 D363 1B              DCX   D
321 D364 CD84D3          CALL  :D384      Read next byte, update c.s
322 D367 DA7ED3          JC    :D37E      If loading error 3
323 D36A 77              MOV   M,A        Store byte in buffer
324 D36B 23              INX   H
325 D36C C35ED3          JMP   :D35E      Next byte
326
327                * If whole block read:
328
329 D36F CDD4D4    LBK20 CALL  :D4D4      Read checksum block contents
330 D372 DA7ED3          JC    :D37E      If loading error 3
331 D375 B8              CMP   B          Check checksum
332 D376 3E02            MVI   A,:02      Loading error 2
333 D378 C280D3          JNZ   :D380      If loading error 2
334 D37B C3B4C6          JMP   :C6B4      CY=1, return: no error
335
336                * If loading error:
337
338 D37E 3E03      LBK40 MVI   A,:03      Loading error 3
339 D380 B7        LOERR ORA   A
340 D381 C3B6C6          JMP   :C6B6      Return with CY=0: error
341                *
342                ************************************
343                * READ BYTE, CALCULATE CHECKSUM *
344                ************************************
345                *
346                * Entry: B: Checksum.
347                * Exit:  A: Byte read.
348                *        B: Updated checksum.
349                *        CDEHL preserved.
350                *
351 D384 CDD4D4    INSC  CALL  :D4D4      Read byte
352 D387 F5        RBUEX PUSH  PSW
353 D388 A8              XRA   B          Calculate checksum
354 D389 07              RLC
355 D38A 47              MOV   B,A        Store new value
356 D38B F1              POP   PSW
357 D38C C9              RET
358                *
359                **********************
360                * READ NAME LENGTH *
361                **********************
362                *
363                * Entry: No conditions.
364                * Exit:  HL: Length name read.
365                *        A:  Result checksum check (0 if OK).
366                *        BCDE: preserved.
367                *        CY=0: O.K.; CY=1: Out of data.
368                *
369 D38D C5        INLNG PUSH  B
370 D38E 0656            MVI   B,:56      Init. checksum
371 D390 CD84D3          CALL  :D384      Read highest length byte
372                                         and update checksum
373 D393 67              MOV   H,A
```

```
374 D394 D484D3             CNC   :D384     Read lowest length byte
375                                         and update checksum
376 D397 6F                 MOV   L,A
377 D398 D4D4D4             CNC   :D4D4     Read checksum on length
378 D39B F5         RHLEX   PUSH  PSW
379 D39C 90                 SUB   B         Check checksum
380 D39D 47                 MOV   B,A
381 D39E F1                 POP   PSW
382 D39F 78                 MOV   A,B
383 D3A0 C1                 POP   B
384 D3A1 C9                 RET
385                 *
386                 ******************************************
387                 * READ + CHECK PROGRAM NAME AND FILE TYPE *
388                 ******************************************
389                 *
390                 * Routine searches for proper file name by
391                 * reading file name and compare it with name
392                 * requested.
393                 *
394                 * Entry: A:  Evt. difference in file type byte
395                 *            read and requested.
396                 *        B:  Requested file type.
397                 *        C:  00 during run program, else FF.
398                 *        DE: Length requested.
399                 *        HL: Address 1st byte name requested.
400                 * Exit:  BCDEHL preserved.
401                 *        A=0: All OK.
402                 *        A=1: Loading error 1.
403                 *
404 D3A2 C5            CMBLK  PUSH  B         Save file type + RUN flag
405 D3A3 E5                   PUSH  H         Save addr reqd name
406 D3A4 47                   MOV   B,A       Store deviation file type
407 D3A5 D5            MBK10  PUSH  D         Save req. name length
408 D3A6 E5                   PUSH  H
409 D3A7 CD8DD3               CALL  :D38D     Read + check program name
410                                           evt. c.s.failure in A
411 D3AA DAEDD3               JC    :D3ED     If reading error
412 D3AD B7                   ORA   A
413 D3AE C2EDD3               JNZ   :D3ED     If checksum error
414 D3B1 E5                   PUSH  H         Save length name on tape
415 D3B2 CD1ADE               CALL  :DE1A     Calculate difference name
416                                           lengths reqd and on tape
417 D3B5 7C                   MOV   A,H
418 D3B6 B5                   ORA   L
419 D3B7 67                   MOV   H,A       Difference in H
420 D3B8 68                   MOV   L,B       Difference file type in L
421 D3B9 D1                   POP   D         Get length name on tape
422 D3BA 0656                 MVI   B,:56     Initiate checksum
423 D3BC E3            MBK20  XTHL            Get byte reqd name
424 D3BD 7A                   MOV   A,D
425 D3BE B3                   ORA   E         Length name on tape = 0 ?
426 D3BF CAD8D3               JZ    :D3D8     If length = 0, or whole
427                                           name read.
428 D3C2 1B                   DCX   D
429 D3C3 CD84D3               CALL  :D384     Read bytes of name, update
430                                           checksum
431 D3C6 DAEDD3               JC    :D3ED     If reading error
432 D3C9 0D                   DCR   C
433 D3CA 0C                   INR   C         Load during run?
434 D3CB F5                   PUSH  PSW       Save length name on tape
435 D3CC C4EBD7               CNZ   :D7EB     Display program name
```

```
436 D3CF F1                 POP   PSW       Get byte of name on tape
437 D3D0 AE                 XRA   M         Compare with name reqd
438 D3D1 23                 INX   H
439 D3D2 E3                 XTHL            Get 'difference flag'
440 D3D3 B4                 ORA   H         Update it
441 D3D4 67                 MOV   H,A       and store it in H
442 D3D5 C3BCD3             JMP   :D3BC     Next byte
443
444               * If whole name read:
445
446 D3D8 CDD4D4  MBK30       CALL  :D4D4     Read c.s on name contents
447 D3DB DAEDD3             JC    :D3ED     If reading error
448 D3DE A8      MBEX        XRA   B         Check checksum
449 D3DF E1                 POP   H
450 D3E0 B5                 ORA   L         Check file type
451 D3E1 6F                 MOV   L,A
452 D3E2 D1                 POP   D         Get length req. name
453 D3E3 7A                 MOV   A,D
454 D3E4 B3                 ORA   E         No name requested ?
455 D3E5 CAE9D3             JZ    :D3E9     If load without name
456 D3E8 7C                 MOV   A,H       Difference in names?
457 D3E9 B5      MBK40      ORA   L         Take also other checks in
458                                         account
459 D3EA E1      MBK45      POP   H
460 D3EB C1                 POP   B
461 D3EC C9                 RET
462
463               * If error:
464
465 D3ED E1      MBK50      POP   H
466 D3EE D1                 POP   D
467 D3EF 3E01               MVI   A,:01     Loading error 1
468 D3F1 C3E9D3             JMP   :D3E9
469               *
470               *
471               *
472 D3F4                    END
```

```
*****************************
* S Y M B O L   T A B L E *
*****************************
```

CHK10	D2C3	CHK20	D2E6	CHK30	D2EB	CMBLK	D3A2
CRBLK	D340	CROPEN	D325	CWBLK	D2F1	CWOPEN	D2B8
INLNG	D38D	INSC	D384	LBK10	D35E	LBK20	D36F
LBK40	D37E	LD34	D2F9	LD35	D307	LD36	D30F
LD37	D316	LOERR	D380	MBEX	D3DE	MBK10	D3A5
MBK20	D3BC	MBK30	D3D8	MBK40	D3E9	MBK45	D3EA
MBK50	D3ED	RBUEX	D387	RCHECK	D2C3	RHLEX	D39B
RLD10	D289	RLEAR	D2B3	RLERR	D2A8	RLOAD	D270
RPN10	D329	RSAVE	D23D				

```
002                             ORG   :D3F4
003                     *
004                     *
005                     *
006                     **********************
007                     * READ FILE HEADER *
008                     **********************
009                     *
010                     * Locates a file on tape and reads leader.
011                     * Exit: Interrupts are disabled. BCDEHL preserved.
012                     *
013 D3F4 CD8FD9         RHDR    CALL  :D98F     Disable sound interrupt
014 D3F7 CD80D4                 CALL  :D480     Find sync pattern
015 D3FA CDD4D4                 CALL  :D4D4     Read flag type byte
016 D3FD DAF4D3                 JC    :D3F4     Again if reading error
017 D400 FE55                   CPI   :55
018 D402 C2F4D3                 JNZ   :D3F4     Again if not flag byte
019 D405 CDD4D4                 CALL  :D4D4     Read file type byte
020 D408 DAF4D3                 JC    :D3F4     Again if reading error
021 D40B C9                     RET
022                     *
023                     ***********************
024                     * WRITE FILE LEADER *
025                     ***********************
026                     *
027                     * Writes a leader for program or data block on tape.
028                     * Disables interrupts which could cause problems.
029                     *
030                     * Entry: at WHDR:  Entry if not during run program.
031                     *        at WHD20: If during run of program.
032                     * Exit:  BCDEHL preserved.
033                     *
034 D40C CDFFDA         WHDR    CALL  :DAFF     Print 'SET RECORD, START
035 D40F 9CDB                   DBL   :DB9C     TAPE, TYPE SPACE'
036 D411 CDCBD7                 CALL  :D7CB     Wait for spacebar pressed
037 D414 CD2ED4         WHD20   CALL  :D42E     Switch on cassette motors
038 D417 F3                     DI              Disable interrupts
039 D418 00                     NOP
040 D419 00                     NOP
041 D41A CDEDD4                 CALL  :D4ED     Write leader
042 D41D 3E55                   MVI   A,:55     Get flag byte
043 D41F C309D5                 JMP   :D509     Write flag byte
044                     *
045                     **************
046                     * (Not used) *
047                     **************
048                     *
049 D422 CDF1D2         MPT27   CALL  :D2F1     Write block on tape
050 D425 00                     NOP
051 D426 00                     NOP
052                     *
053                     ************************
054                     * WRITE FILE TRAILER *
055                     ************************
056                     *
057                     * Write a trailer for program or datablock.
058                     *
059                     * Entry: Length of trailer in C.
060                     * Exit:  A=0, BCDEHL preserved.
061                     *
062                     WTRL
063 D427 CD50D5         CWCLOS  CALL  :D550     Write trailer bytes
```

```
064 D42A FB                 EI              Enable interrupts
065 D42B C345D4             JMP   :D445     Stop cassette motors
066                 *
067                 ************************
068                 * START CASSETTE MOTORS *
069                 ************************
070                 *
071                 * Turns on motor of selected cassettedeck and
072                 * waits 665 msec.
073                 *
074                 * Exit: All registers preserved.
075                 *
076 D42E F5          CASST   PUSH  PSW
077 D42F 3A4000            LDA   :0040     Load POROM
078 D432 F630              ORI   :30       Disable cassette motors
079 D434 E5                PUSH  H
080 D435 213D01            LXI   H,:013D   Addr CASSL
081 D438 AE                XRA   M         Get selected cassette
082 D439 E1                POP   H
083 D43A 324000            STA   :0040     Remember POROM
084 D43D 3206FD            STA   :FD06     Switch cassette motor on
085 D440 CD41DE            CALL  :DE41     Delay
086 D443 F1                POP   PSW
087 D444 C9                RET
088                 *
089                 ************************
090                 * STOP CASSETTE MOTORS *
091                 ************************
092                 *
093                 * Switches off cassettemotors.
094                 *
095                 * Exit: All registers preserved.
096                 *
097                 CRCLOS
098 D445 F5          CASSP   PUSH  PSW
099 D446 3A4000            LDA   :0040     Load POROM
100 D449 F630              ORI   :30       Disable cassette motors
101 D44B 324000            STA   :0040     Remember POROM
102 D44E 3206FD            STA   :FD06     Switch cassette motors off
103 D451 F1                POP   PSW
104 D452 C9                RET
105                 *
106                 ************
107                 * READ BIT *
108                 ************
109                 *
110                 * Reads one bit from tape.
111                 *
112                 * Entry: Address input port in HL. Input low state.
113                 * Exit:  CY=0: sign bit of A is bit read.
114                 *        CY=1: reading error.
115                 *        EHL preserved.
116                 *
117 D453 AF          RBIT    XRA   A
118 D454 57                MOV   D,A
119 D455 47                MOV   B,A
120 D456 4F                MOV   C,A
121
122                 * 1st impulse:
123
124 D457 05          RBT10   DCR   B
125 D458 CA7ED4            JZ    :D47E     Too long low
```

```
126 D45B B6                 ORA   M
127 D45C F257D4             JP    :D457     Wait for high
128 D45F 0D         RBT30   DCR   C
129 D460 CA7ED4             JZ    :D47E     Too long high
130 D463 15                 DCR   D
131 D464 A6                 ANA   M
132 D465 FA5FD4             JM    :D45F     Wait low
133 D468 010000             LXI   B,:0000
134
135                 * 2nd impulse:
136
137 D46B 05         RBT40   DCR   B
138 D46C CA7ED4             JZ    :D47E     Too long low
139 D46F B6                 ORA   M
140 D470 F26BD4             JP    :D46B     Wait high again
141 D473 0D         RBT50   DCR   C
142 D474 CA7ED4             JZ    :D47E     Too long high
143 D477 14                 INR   D
144 D478 A6                 ANA   M
145 D479 FA73D4             JM    :D473     Wait low
146 D47C 7A                 MOV   A,D
147 D47D C9                 RET
148
149                 * If error:
150
151 D47E 37         RBT90   STC               Set CY if error
152 D47F C9                 RET
153                 *
154                 ***************
155                 * READ LEADER *
156                 ***************
157                 *
158                 * Finds a section of leader on the tape.
159                 *
160                 * Entry: No conditions.
161                 * Exit:  BCDEHL preserved, interrupts disabled.
162                 *
163 D480 C5          RLEAD   PUSH  B
164 D481 D5                 PUSH  D
165 D482 E5                 PUSH  H
166 D483 0628               MVI   B,:28       Estimate of impulse length
167 D485 2100FD             LXI   H,:FD00     address input port
168 D488 3EFF       RDL05   MVI   A,:FF
169 D48A FB         RDL10   EI                Enable interrupts
170 D48B 00                 NOP               (here cursor flashes)
171 D48C F3                 DI                Disable interrupts
172 D48D A6                 ANA   M
173 D48E FA8AD4             JM    :D48A       Wait low
174 D491 48                 MOV   C,B         Estimated length in C
175 D492 1614               MVI   D,:14       Needs this many cycles for
176                                           synchronisation. Must be
177                                           more than trailer length.
178 D494 1E00       RDL30   MVI   E,:00
179 D496 AF                 XRA   A
180 D497 1D         RDL40   DCR   E
181 D498 CA88D4             JZ    :D488       Too long low; start again
182 D49B B6                 ORA   M
183 D49C F297D4             JP    :D497       Wait high
184 D49F 0600               MVI   B,:00
185 D4A1 04         RDL50   INR   B
186 D4A2 CA88D4             JZ    :D488       Too long high; start again
187 D4A5 A6                 ANA   M
```

```
188 D4A6 FAA1D4             JM    :D4A1     Wait low
189 D4A9 78                 MOV   A,B
190 D4AA 91                 SUB   C         Compare impulse length
191                                         with estimate
192 D4AB F2B0D4             JP    :D4B0
193 D4AE 2F                 CMA             )
194 D4AF 3C                 INR   A         ) 2-complement if <0
195 D4B0 5F         RDL60   MOV   E,A       Store difference in E
196 D4B1 79                 MOV   A,C       Calculate margin
197 D4B2 E6F0               ANI   :F0
198 D4B4 1F                 RAR
199 D4B5 1F                 RAR
200 D4B6 1F                 RAR             Margin: 1/8th of estimate
201 D4B7 BB                 CMP   E         Compare with difference
202 D4B8 DAC3D4             JC    :D4C3     Not within margin
203
204                 * If sync archieved:
205
206 D4BB 15                 DCR   D
207 D4BC C294D4             JNZ   :D494     Next impulse until D=0
208 D4BF 14                 INR   D
209 D4C0 C394D4             JMP   :D494     Next impulse until out
210                                         of margin
211
212                 * If out of margin:
213
214 D4C3 15         RDL70   DCR   D
215 D4C4 C288D4             JNZ   :D488     Not synchronised; again
216 D4C7 AF                 XRA   A
217 D4C8 B6         RDL80   ORA   M
218 D4C9 F2C8D4             JP    :D4C8     Wait high
219 D4CC A6         RDL90   ANA   M
220 D4CD FACCD4             JM    :D4CC     Wait low
221 D4D0 E1                 POP   H
222 D4D1 D1                 POP   D
223 D4D2 C1                 POP   B
224 D4D3 C9                 RET
225                 *
226                 *************
227                 * READ BYTE *
228                 *************
229                 *
230                 * Reads one byte from tape.
231                 *
232                 * Entry: No conditions.
233                 * Exit:  CY=0: Byte read in A.
234                 *        CY=1: Some error.
235                 *        BCDEHL preserved.
236                 *
237 D4D4 C5           RBYTE  PUSH  B
238 D4D5 D5                  PUSH  D
239 D4D6 E5                  PUSH  H
240 D4D7 2100FD              LXI   H,:FD00   Address input port
241 D4DA 1EFE                MVI   E,:FE
242 D4DC CD53D4      RBY10   CALL  :D453     Read bit
243 D4DF DAE9D4             JC    :D4E9     If reading error: CY=1
244 D4E2 17                 RAL
245 D4E3 7B                 MOV   A,E
246 D4E4 17                 RAL
247 D4E5 5F                 MOV   E,A       Shift bit into E
248 D4E6 DADCD4             JC    :D4DC     Next bit
249 D4E9 E1          RBY20  POP   H         8 bits read, no error
```

```
D4EA D1                 POP   D
D4EB C1                 POP   B
D4EC C9                 RET
                *
                ****************
                * WRITE LEADER *
                ****************
                *
                * Writes a leader on the tape. From WLD10 also used
                * to write a trailer.
                *
                * Entry: No conditions.
                * Exit:  A=0, BCDEHL preserved.
                *
D4ED 00         WLEAD   NOP
D4EE C5                 PUSH  B
D4EF E5                 PUSH  H
D4F0 2AE602             LHLD  :02E6     Get leader impulse length
D4F3 01E807             LXI   B,:07E8   Period for synchr.
D4F6 CD24D5     WLD10   CALL  :D524     Write bit
D4F9 0B                 DCX   B
D4FA 78                 MOV   A,B
D4FB B1                 ORA   C
D4FC C2F6D4             JNZ   :D4F6     Write many bits
D4FF 2AE802             LHLD  :02E8     Get impulse length data bit
D502 CD24D5             CALL  :D524     Write a data '1' bit to end
D505 E1                 POP   H
D506 C1                 POP   B
D507 00                 NOP
D508 C9                 RET
                *
                ***************
                * WRITE BYTE *
                ***************
                *
                * Write a byte to tape.
                *
                * Entry: Byte to be written in A.
                * Exit:  All registers preserved.
                *
D509 F5         WBYTE   PUSH  PSW
D50A C5                 PUSH  B
D50B D5                 PUSH  D
D50C E5                 PUSH  H
D50D 2AE802             LHLD  :02E8     Get impulse length bit '1'
D510 5C                 MOV   E,H
D511 55                 MOV   D,L       DE: inpulse length bit '0'
D512 0608               MVI   B,:08     8 bits to write
D514 17         WBY10   RAL             Set/reset CY for kind of bit
D515 DC24D5             CC    :D524     Write data '1' bit
D518 EB                 XCHG
D519 D424D5             CNC   :D524     Write data '0' bit
D51C EB                 XCHG
D51D 05                 DCR   B
D51E C214D5             JNZ   :D514     Next bit
D521 C356CB             JMP   :CB56     Pop all, ret
                *
                *************
                * WRITE BIT *
                *************
                *
                * Write 2 impulses on tape, one long, one short.
```

```
312                     *
313                     * Entry: H: Half count first cycle.
314                     *        L: Half count second cycle.
315                     * Exit:  All registers preserved.
316                     *
317 D524 F5             WBIT    PUSH  PSW
318 D525 D5                     PUSH  D
319 D526 E5                     PUSH  H
320 D527 6C                     MOV   L,H
321 D528 1106FD                 LXI   D,:FD06   Address output port
322 D52B CD3CD5                 CALL  :D53C     Write 1st impulse
323 D52E E1                     POP   H
324 D52F E5                     PUSH  H
325 D530 65                     MOV   H,L
326 D531 7D                     MOV   A,L
327 D532 D608                   SUI   :08       Allow for return to WBYTE
328 D534 6F                     MOV   L,A
329 D535 CD3CD5                 CALL  :D53C     Write 2nd impulse
330 D538 E1                     POP   H
331 D539 D1                     POP   D
332 D53A F1                     POP   PSW
333 D53B C9                     RET
334                     *
335                     ***************
336                     * WRITE CYCLE *
337                     ***************
338                     *
339                     * Writes one impulse (hi/lo) on tape. Two cycles
340                     * are required for one bit.
341                     *
342                     * Entry: DE: Address output port.
343                     *        HL: Impulse length constants.
344                     * Exit:  HL = 0. BCDE preserved.
345                     *
346 D53C 3A4000          WCYC    LDA   :0040     POROM in A
347 D53F F601                   ORI   :01       lsb = 1
348 D541 12                     STAX  D         Output port is made '1'
349 D542 25             WCY10   DCR   H
350 D543 C242D5                 JNZ   :D542     Write '1' until H=0
351 D546 2D                     DCR   L
352 D547 2D                     DCR   L
353 D548 2D                     DCR   L         Allow for return to WBIT
354 D549 3D                     DCR   A
355 D54A 12                     STAX  D         Output port is made '0'
356 D54B 2D             WCY20   DCR   L
357 D54C C24BD5                 JNZ   :D54B     Write '0' until L=0
358 D54F C9                     RET
359                     *
360                     ***********************
361                     * WRITE TRAILER BITS *
362                     ***********************
363                     *
364                     * Writes trailer bits after a block on tape.
365                     *
366                     * Entry: Number of trailer bits in C.
367                     * Exit:  A=0, other registers preserved.
368                     *        F corrupted.
369                     *
370 D550 C5             WTRLX   PUSH  B
371 D551 E5                     PUSH  H
372 D552 2AEA02                 LHLD  :02EA     Trailer impulse length
373 D555 0600                   MVI   B,:00
```

```
374 D557 C3F6D4             JMP   :D4F6     Write trailer bits
375                  *
376 D55A FF                 DATA  :FF
377 D55B FF                 DATA  :FF
378 D55C FF                 DATA  :FF
379 D55D FF                 DATA  :FF
380 D55E FF                 DATA  :FF
381 D55F FF                 DATA  :FF
382                  *
383                  ********************************
384                  * INITIALISE KEYBOARD POINTERS *
385                  ********************************
386                  *
387                  * Set all keyboard pointers to default values.
388                  *
389                  * Entry: Address ASCII-table in HL (3E8C5).
390                  * Exit:  BCDE preserved.
391                  *
392                  KLIRS
393 D560 22A702       KBINIT SHLD  :02A7     Load pointer ASCII-table
394 D563 AF           KLIRP  XRA   A
395 D564 32B902             STA   :02B9     Allow complete scan routine
396 D567 32C302             STA   :02C3     CTRL not pressed
397 D56A 21BA02             LXI   H,:02BA
398 D56D 22BE02             SHLD  :02BE     Set KLIIN ) Ignore
399 D570 22C002             SHLD  :02C0     Set KLIOU ) previous inputs
400 D573 2F                 CMA
401 D574 32C402             STA   :02C4     BREAK pointer = FF
402 D577 C9                 RET
403                  *
404                  ****************************************
405                  * KEYBOARD INTERRUPT SERVICE (RST 6) *
406                  ****************************************
407                  *
408                  * Current interrupt mask is saved. Only stack and
409                  * clock interrupts are allowed. Keyboard timer 4
410                  * is re-loaded.
411                  * KBXCT is counted down: abort if not 0.
412                  * Else: scan keyboard and store result.
413                  *
414                  * Entry: None.
415                  * Exit:  All registers + int. mask preserved.
416                  *
417 D578 F5           KBINT  PUSH  PSW
418 D579 C5                  PUSH  B
419 D57A D5                  PUSH  D
420 D57B 3A5F00              LDA   :005F
421 D57E F5                  PUSH  PSW      Preserve current int. mask
422 D57F 3E84                MVI   A,:84    )
423 D581 32F8FF              STA   :FFF8    ) Allow stack and clock
424 D584 325F00              STA   :005F    ) interrupts only
425 D587 FB                  EI
426 D588 CD9DD9              CALL  :D99D    Reload keyboard timer
427 D58B 21C101              LXI   H,:01C1
428 D58E 35                  DCR   M        Decr. keyb.scan time count
429 D58F C2CDD9              JNZ   :D9CD    No scanning if <>0
430
431                  * if KBXCT = 0:
432
433 D592 3602                MVI   M,:02    Set keyb. scan time counter
434 D594 CD9AD5              CALL  :D59A    Scan keyboard, store result
435 D597 C3CDD9              JMP   :D9CD    Restore int.mask; ret.
```

```
436                     *
437                     ********************************
438                     * SCAN KEYBOARD, STORE RESULT *
439                     ********************************
440                     *
441                     * Exit: All registers corrupted.
442                     *
443 D59A 11F1FF     KBSCAN  LXI   D,:FFF1   Input port from keyboard
444 D59D 21F7FF             LXI   H,:FFF7   Output port to keyboard
445 D5A0 01C402             LXI   B,:02C4   BREAK pointer
446 D5A3 F3                DI
447 D5A4 3640              MVI   M,:40     Scan row 6
448 D5A6 1A                LDAX  D         and get result
449 D5A7 FB                EI
450 D5A8 87                ADD   A         Check for BREAK pressed
451 D5A9 FA06D6            JM    :D606     If BREAK pressed
452 D5AC CD50D7            CALL  :D750     Update BREAK pointer
453 D5AF 3AB902            LDA   :02B9     Get BREAK pointer
454 D5B2 B7                ORA   A         Scan for BREAK only?
455 D5B3 C205D6            JNZ   :D605     Then abort
456
457                     * Scan all rows and store result in MAP1:
458
459 D5B6 01A902            LXI   B,:02A9   MAP1 for currently
460                                         pressed key
461 D5B9 C5                PUSH  B         Preserve MAP1 addr
462 D5BA 3C                INR   A         Determine row
463 D5BB F5         KEB10  PUSH  PSW
464 D5BC F3                DI
465 D5BD 77                MOV   M,A       Scan row
466 D5BE 1A                LDAX  D         Get result
467 D5BF FB                EI
468 D5C0 02                STAX  B         Store result in MAP1
469 D5C1 03                INX   B
470 D5C2 F1                POP   PSW
471 D5C3 87                ADD   A         Determine next row
472 D5C4 D2BBD5            JNC   :D5BB     Scan next row if not ready
473
474                     * REPT handling:
475
476 D5C7 3AAF02            LDA   :02AF
477 D5CA E620              ANI   :20       Check if REPT pressed
478 D5CC 47                MOV   B,A       Store result
479 D5CD 21C202            LXI   H,:02C2   Addr. REPT counter
480 D5D0 7E                MOV   A,M       Get contents
481 D5D1 3601              MVI   M,:01     Update it for immediate scan
482 D5D3 CADFD5            JZ    :D5DF     If REPT not pressed
483 D5D6 3D                DCR   A         Else
484 D5D7 77                MOV   M,A       Decr. REPT counter
485 D5D8 C204D6            JNZ   :D604     If <>0, abort scan
486 D5DB 3602              MVI   M,:02     Else RPCNT=2
487 D5DD 06FF              MVI   B,:FF     Set B=FF for REPT pressed
488
489                     * ASCII-value of key pressed into KLIND:
490
491 D5DF E1           KEB40  POP   H         Get addr MAP1
492 D5E0 E5                PUSH  H         Save addr. MAP1
493 D5E1 11B102            LXI   D,:02B1   Get addr. MAP2
494 D5E4 0E00              MVI   C,:00
495 D5E6 7E           KEB50  MOV   A,M       Get result scan current
496                                         row in A
497 D5E7 05                DCR   B
```

```
498 D5E8 04                    INR   B        REPT pressed ?
499 D5E9 C2F0D5                JNZ   :D5F0    If REPT pressed
500 D5EC EB                    XCHG
501 D5ED AE                    XRA   M        ) Check if new input
502 D5EE EB                    XCHG           )
503 D5EF A6                    ANA   M        )
504 D5F0 B7          KEB60     ORA   A        )
505 D5F1 C432D6                CNZ   :D632    If new: Get ASCII-code
506                                           and store it in KLIND
507 D5F4 13                    INX   D
508 D5F5 23                    INX   H        Next row
509 D5F6 0C                    INR   C
510 D5F7 79                    MOV   A,C
511 D5F8 FE08                  CPI   :08      All rows checked?
512 D5FA C2E6D5                JNZ   :D5E6    Next row if not
513 D5FD D1          KEB70     POP   D        Get MAP1 addr in DE
514 D5FE D5                    PUSH  D
515 D5FF 44                    MOV   B,H      ) Get MAP2 addr in HL
516 D600 4D                    MOV   C,L      )
517 D601 CD4FDE                CALL  :DE4F    Transfer (MAP1) into MAP2
518 D604 D1          KEB80     POP   D        Scrap
519 D605 C9          KEB81     RET
520
521                  * if BREAK pressed:
522
523 D606 C5          KEB90     PUSH  B        Save Breakpntr
524 D607 CDA6D8                CALL  :D8A6    All sound off
525 D60A C1                    POP   B
526 D60B 00                    NOP
527 D60C CD45D4                CALL  :D445    Stop cassette motors
528 D60F 0A                    LDAX  B        Get KBRFL
529 D610 3C                    INR   A
530 D611 CA05D6                JZ    :D605    If KBRFL=FF: break acknow-
531                                           ledged already
532 D614 02                    STAX  B        Else: store new KBRFL
533 D615 FE20                  CPI   :20
534 D617 C205D6                JNZ   :D605    If new KBRFL<>20: wait for
535                                           soft-break to be accepted
536 D61A CD63D5                CALL  :D563    Else: init keyb. pointers
537 D61D C30CC8                JMP   :C80C    Print 'BREAK', return to
538                                           monitor
539                  *
540                  *
541 D620                       END
```

```
****************************
* S Y M B O L   T A B L E *
****************************
```

CASSP	D445	CASST	D42E	CRCLOS	D445	CWCLOS	D427
KBINIT	D560	KBINT	D578	KBSCAN	D59A	KEB10	D5BB
KEB40	D5DF	KEB50	D5E6	KEB60	D5F0	KEB70	D5FD
KEB80	D604	KEB81	D605	KEB90	D606	KLIRP	D563
KLIRS	D560	MPT27	D422	RBIT	D453	RBT10	D457
RBT30	D45F	RBT40	D46B	RBT50	D473	RBT90	D47E
RBY10	D4DC	RBY20	D4E9	RBYTE	D4D4	RDL05	D488
RDL10	D48A	RDL30	D494	RDL40	D497	RDL50	D4A1
RDL60	D4B0	RDL70	D4C3	RDL80	D4C8	RDL90	D4CC
RHDR	D3F4	RLEAD	D480	WBIT	D524	WBY10	D514
WBYTE	D509	WCY10	D542	WCY20	D54B	WCYC	D53C
WHD20	D414	WHDR	D40C	WLD10	D4F6	WLEAD	D4ED
WTRL	D427	WTRLX	D550				

```
002                             ORG   :D620
003                     *
004                     *
005                     *
006                     ****************************
007                     * COMPLETE KEYBOARD SCAN *
008                     ****************************
009                     *
010                     * Initialises a complete keyboard scan,
011                     * independent of the KNSCAN flag, and performs it.
012                     *
013                     * Exit: All registers preserved.
014                     *
015 D620 F5             KFSCAN  PUSH  PSW
016 D621 C5                     PUSH  B
017 D622 D5                     PUSH  D
018 D623 E5                     PUSH  H
019 D624 21B902                 LXI   H,:02B9   Addr KNSCAN pointer
020 D627 3600                   MVI   M,:00     Enable complete scan
021 D629 E5                     PUSH  H
022 D62A CD9AD5                 CALL  :D59A     Scan keyboard and store
023                                             result in circ.buffer
024 D62D E1                     POP   H
025 D62E 35                     DCR   M         Scan for BREAK only
026 D62F C356CB                 JMP   :CB56     Popall; return
027                     *
028                     ************************************
029                     * GET ASCII VALUE OF KEY PRESSED *
030                     ************************************
031                     *
032                     * Calculates address in ASCII table in ROM and gets
033                     * ASCII value of the key pressed. The result is
034                     * stored in the 4-bye circular buffer KLIND.
035                     *
036                     * Entry: A:  Keycode of scanned row (7 bits only).
037                     *        B:  FF when REPT pressed; else 00.
038                     *        C:  Number of row.
039                     *        DE: Address in MAP2.
040                     *        HL: Address in MAP1.
041                     * Exit:  BCDEHL preserved; AF corrupted.
042                     *
043 D632 C5             TKEY    PUSH  B
044 D633 0607                   MVI   B,:07     Check which key in row is
045 D635 1F             TKY10   RAR             pressed; calculate offset
046 D636 DC3FD6                 CC    :D63F     Get ASCII value; store it
047                                             in KLIND
048 D639 05                     DCR   B
049 D63A C235D6                 JNZ   :D635     Next column
050 D63D C1                     POP   B
051 D63E C9                     RET
052                     *
053                     * GET KEY-ASCII VALUE AND STORE IT:
054                     *
055 D63F CF             SINKEY  RST   1         Get ASCII-value from ROM-
056 D640 12                     DATA  :12       table and store it in KLIND
057 D641 C9                     RET
058                     *
059                     ********************************
060                     * OUTPUT TO RS232 IF REQUIRED *
061                     ********************************
062                     *
063                     * Checks if output is to RS232. If positive,
```

```
                        * output is performed.
                        *
                        * Entry: Byte to be transmitted in A.
                        *
D642 F5                 TOUTSE  PUSH  PSW
D643 3A3101                     LDA   :0131     Get output direction
D646 B7                         ORA   A         Check if RS232 output
D647 C28CDD                     JNZ   :DD8C     Abort if not
D64A F1                         POP   PSW
D64B C394DD                     JMP   :DD94     Output to RS232
                        *
                        ******************************************
                        * GET ASCII-VALUE OF CHARACTER IN BUFFER *
                        ******************************************
                        *
                        * Routine is not used.
                        *
                        * The Ascii-value of a character is stored in
                        * KLIND. Afterwards, Bank select is restored.
                        *
D64E CD3FE9             LD67    CALL  :E93F     (3) Ascii-value in KLIND
D651 F1                         POP   PSW       A contains POROM
D652 CD08D8                     CALL  :D808     Update PORO and POROM
D655 F1                         POP   PSW
D656 E1                         POP   H
D657 C9                         RET
                        *
                        ************************************
                        * parts of RUN 'RANDOMISE' (0E40C) *
                        ************************************
                        *
                        RMI15
D658 3D                 MPT39   DCR   A
D659 C258D6                     JNZ   :D658     Again till A=0
D65C 7E                         MOV   A,M       Get contents FD00
D65D AB                         XRA   E
D65E C9                         RET
                        *
                        * Entry: L = 0.
                        *
D65F 3AC101             MPT38   LDA   :01C1     Get keyb.scan time count
                                                (0, 1 or 2)
D662 0F                         RRC
D663 0F                         RRC             A= 0, #40 or #80
D664 5F                         MOV   E,A       in E
D665 45                         MOV   B,L       )
D666 4D                         MOV   C,L       ) BC=0
D667 C9                         RET
                        *
                        ************************************
                        * WRITE 2 BLOCKS + TRAILER ON TAPE *
                        ************************************
                        *
                        * Entry: HL:    Startaddress 1st block.
                        *        Stack: Length 2nd block.
                        *               Startaddress 2nd block.
                        *
D668 110100             MPT13   LXI   D,:0001   Length 1st block = 1
D66B CDC802                     CALL  :02C8     Write 1st block
D66E D1                         POP   D         Get length 2nd block
D66F E1                         POP   H         Get startaddr. 2nd block
D670 CDC802                     CALL  :02C8     Write 2nd block
```

```
126 D673 CDCB02              CALL  :02CB     Write trailer
127 D676 B7                  ORA   A
128 D677 C9                  RET
129                     *
130                     ***********************************
131                     * LOADA: EVALUATE PROGRAM NAME *
132                     ***********************************
133                     *
134                     * The program name is evaluated. Selection of
135                     * ROM bank 1 is prepared.
136                     *
137 D678 CD87D6      MPT14   CALL  :D687     Evaluate program name
138 D67B 3A4000              LDA   :0040     Get POROM
139 D67E F640                ORI   :40       Prepare selection ROMbank 1
140 D680 C9                  RET
141                     *
142                     ********************
143                     * OPEN READ FILE *
144                     ********************
145                     *
146 D681 D5          MPT18   PUSH  D
147 D682 CDCE02              CALL  :02CE     Open READ file
148 D685 D1                  POP   D
149 D686 C9                  RET
150                     *
151                     ************************************
152                     * EVALUATE A STRING EXPRESSSION *
153                     ************************************
154                     *
155                     * A string expression is evaluated. Eventually,
156                     * the Heap entry is cleared if the string was
157                     * temporarily on Heap.
158                     *
159                     * Exit: DE preserved, BC updated.
160                     *       HL points after string
161                     *
162 D687 D5          MPT15   PUSH  D
163 D688 CD91E7              CALL  :E791     (0) Eval. string expr.
164                                          evt. clear Heap entry
165 D68B D1                  POP   D
166 D68C C9                  RET
167                     *
168                     ********************
169                     * CURSOR HANDLING *
170                     ********************
171                     *
172                     * Load the cursor pointers with the address, the
173                     * colour and the contents of a new cursor address.
174                     *
175                     * Entry: HL: New cursor address.
176                     *         D:  The colour byte of this location.
177                     *         E:  The contents of this location.
178                     * Exit:  HL and DE exchanged; ABCF preserved.
179                     *
180 D68D 227200      SPT00   SHLD  :0072     Store cursor address
181 D690 EB                  XCHG
182 D691 227600              SHLD  :0076     Store cursor addr.contents
183 D694 C9                  RET
184                     *
185                     ************************
186                     * OUTPUT ONE CHARACTER *
187                     ************************
```

```
                        *
                        * Output direction depending on OTSW.
                        * This routine is useable for all data output
                        * functions in machine language programs.
                        *
                        * Entry: Character for output in A.
                        * Exit:  All registers preserved.
                        *
D695 F5         MPT31   PUSH  PSW
D696 CD60DD             CALL  :DD60     Output character in A.
D699 F1                 POP   PSW
D69A C9                 RET
                        *
D69B FF                 DATA  :FF
                        *
                        ************************************************
                        * KEYB. SCANNING: UPDATE POINTER OUTPUT BUFFER *
                        ************************************************
                        *
                        * Updates the pointer to the circular output
                        * buffer #02BA-#02BD.
                        *
                        * Entry: HL: KLIOU.
                        * Exit:  HL: Updated KLIOU.
                        *        AF corrupted. BCDE preserved.
                        *
D69C 23         KPTRU   INX   H         Incr. KLIOU
D69D 7D                 MOV   A,L       Lobyte into A
D69E FEBE               CPI   :BE       Buffer full?
D6A0 C0                 RNZ             Quit if not
D6A1 21BA02             LXI   H,:02BA   Else: wrap around
D6A4 C9                 RET
                        *
                        ********************************************
                        * KEYBOARD SCANNING: CHECK IF NEW INPUTS *
                        ********************************************
                        *
                        * Returns a flag if BREAK has been pressed or if
                        * there is a character available.
                        *
                        * Entry: No conditions.
                        * Exit:  BCDEHL preserved.
                        *        A: Difference KLIIN and KLIOU.
                        *        CY=1: Break pressed.
                        *        CY=0, Z=1: No inputs.
                        *        CY=0, Z=0: New input available.
                        *
                        ASKKEY
D6A5 E5         BREAK   PUSH  H

                        * If suspended:

D6A6 21C402             LXI   H,:02C4   Addr break pntr
D6A9 7E                 MOV   A,M
D6AA 3D                 DCR   A
D6AB FEFE               CPI   :FE       Test if not 0 or FF
D6AD DAB9D6             JC    :D6B9     Abort if break: CY=1
                        *
D6B0 2AC002             LHLD  :02C0     Get KLIOU
D6B3 3ABE02             LDA   :02BE     Get KLIIN
D6B6 95                 SUB   L         New keys pressed?
D6B7 37                 STC
```

```
D6B8 3F              CMC              CY=0
D6B9 E1        OTK10 POP   H
D6BA C9              RET
                     *
                     ******************
                     * INPUT SCANNING *
                     ******************
                     *
                     * Gets input from keyboard or DINC, depending on
                     * INSW (0296).
                     *
                     * FGETC: Gets a character, even if keyboard
                     *        scanning is turned off.
                     *  GETC: Returns a flag if break, and sets break
                     *        accepted. Returns also a flag if a
                     *        character is available.
                     *
                     * Exit: CY=1: Break pressed.
                     *       Z=1:  No inputs. Then A=0.
                     *       Else: Character in A.
                     *
D6BB CD20D6    FGETC CALL  :D620      Complete keyboard scan
D6BE C3DBD1    GETC  JMP   :D1DB      Check input keyb/DINC;
                                      scan for new keys pressed
D6C1 DAD4D6    MPR29 JC    :D6D4      Jump if Break pressed.
D6C4 C8              RZ               No new input or buffer full

                     * If inputs:

D6C5 E5              PUSH  H
D6C6 2AC002          LHLD  :02C0      Get addr pntr output buffer
D6C9 7E              MOV   A,M        Get ASCII char in A
D6CA F5              PUSH  PSW
D6CB CD9CD6          CALL  :D69C      Update pntr
D6CE F1              POP   PSW
D6CF 22C002          SHLD  :02C0      Re-instate KLIOU
D6D2 E1              POP   H
D6D3 C9              RET              CY=0

                     * If Break pressed:

D6D4 3EFF      GTC10 MVI   A,:FF      Flag 'break accepted'
D6D6 32C402          STA   :02C4      Scan for break only
D6D9 C9              RET              CY=1
                     *
                     **********************
                     * WAIT FOR SPACEBAR *
                     **********************
                     *
                     * Wait until spacebar (or break) is pressed.
                     *
                     * Entry: None.
                     * Exit:  CY=1: Break pressed.
                     *        CY=0: Space in A.
                     *        BCDEHL preserved.
                     *
D6DA CDBBD6    WSPACE CALL  :D6BB      Input scanning
D6DD D8              RC               Abort if BREAK pressed
D6DE FE20            CPI   :20        Check if spacebar
D6E0 C2DAD6          JNZ   :D6DA      Wait for space bar
D6E3 B7              ORA   A
D6E4 C9              RET
```

```
312                     *
313                     ***************************
314                     * part of LOADA (1EE0F) *
315                     ***************************
316                     *
317 D6E5 E3             MPT20   XTHL            Orig. DE in HL, free RAM
318                                             pntr on stack
319 D6E6 CD14DE                 CALL  :DE14     Compare DE-HL
320 D6E9 D2EDD6                 JNC   :D6ED     If DE<=HL
321 D6EC EB                     XCHG
322 D6ED 42             RLA15   MOV   B,D       ) Lowest value in BC
323 D6EE 4B                     MOV   C,E       )
324 D6EF D1                     POP   D
325 D6F0 E1                     POP   H
326 D6F1 E3                     XTHL
327 D6F2 C34CEE                 JMP   :EE4C     (1) Continu
328                     *
329                     ******************************************
330                     * CHECK SUFFICIENT SCREEN RAM AVAILABLE *
331                     * PREPARE SELECTION SPLIT MODE          *
332                     ******************************************
333                     *
334 D6F5 37             SPT01   STC             CY=1
335 D6F6 2A9000                 LHLD  :0090     Get end area splitting mode
336 D6F9 CDA6E5                 CALL  :E5A6     (2) Ask for temporary area
337 D6FC 21A0E5                 LXI   H,:E5A0   (2) Startaddr table screen
338                                             parameters split modes
339 D6FF C9                     RET
340                     *
341                     ******************************************
342                     * CHANGE FROM SPLIT TO FULL GRAPHIC MODE *
343                     ******************************************
344                     *
345 D700 CDF5D6         SSM20   CALL  :D6F5     Check suff. screen RAM
346 D703 C385E4                 JMP   :E485     (2) Change split to full
347                     *
348                     ******************************************
349                     * CHECK SUFFICIENT SCREEN RAM AVAILABLE *
350                     * PREPARE FULL GRAPHICS MODE            *
351                     ******************************************
352                     *
353 D706 CDF5D6         SPTA2   CALL  :D6F5     Check suff. screen RAM
354 D709 219AE5                 LXI   H,:E59A   (2) Startaddr table screen
355                                             parameters full graph.modes
356 D70C C9                     RET
357                     *
358                     ********************************
359                     * SET UP CURRENT SCREEN MODE *
360                     ********************************
361                     *
362 D70D D1             SMA20   POP   D
363 D70E F1                     POP   PSW
364 D70F 3A9D00         SMA30   LDA   :009D     Get current screen mode
365 D712 B7                     ORA   A
366 D713 1F                     RAR             Split or all-graph mode ?
367 D714 CD39E5                 CALL  :E539     (2) Set up screen mode
368 D717 C33CE4                 JMP   :E43C     (2) Pop PSW, ret
369                     *
370                     ************************
371                     * STOP LOADING PROGRAMS *
372                     ************************
373                     *
```

```
374                     * Entry: HL: New end symbol table.
375                     * Exit:  All registers preserved.
376                     *
377 D71A 22A302         MPT11   SHLD  :02A3      Store end symtab
378 D71D C3D402                 JMP   :02D4      Stop loading
379                     *
380                     ******************************
381                     * INIT. WRITING FILE LEADER *
382                     ******************************
383                     *
384                     * Procedure depends on saving in program or not.
385                     *
386 D720 E5             MPT21   PUSH  H
387 D721 2A0001                 LHLD  :0100      Get start current line
388 D724 7C                     MOV   A,H
389 D725 B5                     ORA   L          0 if not during run
390 D726 E1                     POP   H
391 D727 C214D4                 JNZ   :D414      If SAVE during run
392 D72A C30CD4                 JMP   :D40C      Write file leader
393                     *
394                     ********************************************
395                     * INIT. SOUNDGENERATOR, GIC, START HEAP, *
396                     * MOVE CASSETTE VECTORS, GET DCE INPUTS  *
397                     ********************************************
398                     *
399 D72D CDA6D8         MPT00   CALL  :D8A6      Init. sound generator
400 D730 CD95D7                 CALL  :D795      Transfer cassette vectors
401 D733 21EC02                 LXI   H,:02EC
402 D736 CF                     RST   1          Init. GIC; get evt. inputs
403 D737 0C                     DATA  :0C        from DCE-bus (bootstrap)
404 D738 229B02                 SHLD  :029B      Set HEAP start
405 D73B C9                     RET
406                     *
407 D73C 02                     DATA  :02        (not used)
408                     *
409                     ********************************************
410                     * part of RUN A VARIABLE REFERENCE (0E95A) *
411                     ********************************************
412                     *
413 D73D CD6DE9         MPT49   CALL  :E96D      (0) Run VARPTR
414 D740 D1                     POP   D
415 D741 C9                     RET
416                     *
417 D742 DD                     DATA  :DD        (not used)
418                     *
419                     ****************************************************
420                     * SET INPUT DIRECTION, LOAD SOUND + KEYB TIMERS *
421                     ****************************************************
422                     *
423                     * Part of 'stack interrupt' (D9E2).
424                     *
425 D743 323501         MPT30   STA   :0135      Set input direction
426 D746 CD9DD9                 CALL  :D99D      Reload keyboard timer
427 D749 C3A3D9                 JMP   :D9A3      Reload sound timer, ret
428                     *
429                     ********************************
430                     * DATA OUTPUT ROUTINE 'DOUTC' *
431                     ********************************
432                     *
433                     * Part of DD70.
434                     * On #02DD, an jump to an user determined output
435                     * routine can be written. As default, a RET is
```

```
436                     * on this address.
437                     *
438 D74C F1             OTC30   POP   PSW       Output char in A
439 D74D C3DD02                 JMP   :02DD     Goto user DOUTC
440                     *
441                     **********************
442                     * CHECK BREAK FLAG *
443                     **********************
444                     *
445                     * Part of 'scan keyboard' (D59A).
446                     *
447                     * Entry: Address 'break' flag in BC.
448                     * Exit:  BCDEHL preserved, AF corrupted.
449                     *
450 D750 0A             MPT28   LDAX  B           Get KBRFL
451 D751 3C                     INR   A
452 D752 C0                     RNZ               Quit if it was <> FF
453 D753 02                     STAX  B           KBRFL=0: No recent break
454 D754 C9                     RET
455                     *
456                     ****************************
457                     * SOUND INTERRUPT (RST 3) *
458                     ****************************
459                     *
460                     * Called periodically every few milliseconds.
461                     * Adjust the volume for the sound channels and
462                     * approaches the correct frequency if necessary.
463                     *
464                     * Saves all registers + interrupt mask.
465                     * Enables only clock and sound interrupts.
466                     * Sound interrupt timer is re-loaded and sound
467                     * control blocks are executed.
468                     *
469                     * Entry: HL must already be saved on stack.
470                     * Exit:  All registers preserved.
471                     *        Bank select is restored.
472                     *
473 D755 F5             SNTMP   PUSH  PSW
474 D756 C5                     PUSH  B
475 D757 D5                     PUSH  D
476 D758 3A5F00                 LDA   :005F       Get current int. mask
477 D75B F5                     PUSH  PSW         and save it
478 D75C 3E84                   MVI   A,:84       )
479 D75E 325F00                 STA   :005F       ) Enable clock and sound
480 D761 32F8FF                 STA   :FFF8       ) interrupts only
481 D764 FB                     EI
482 D765 CDA3D9                 CALL  :D9A3       Reload sound timer
483 D768 3A4000                 LDA   :0040       Get POROM
484 D76B F5                     PUSH  PSW         and save it
485 D76C E63F                   ANI   :3F
486 D76E F640                   ORI   :40         Select ROM bank 1
487 D770 324000                 STA   :0040       Set POROM
488 D773 3206FD                 STA   :FD06       and PORO
489 D776 CD6EEE                 CALL  :EE6E       (1) Execute SCB('s)
490 D779 F1                     POP   PSW
491 D77A 324000                 STA   :0040       Re-instate old ROM bank
492 D77D 3206FD                 STA   :FD06       and save it
493 D780 C3CDD9                 JMP   :D9CD       Restore int. mask; ret
494                     *
495                     ************************
496                     * FAILURE DURING ROPEN *
497                     ************************
```

```
498                     *
499 D783 05             MPT54   DCR   B
500 D784 04                     INR   B
501 D785 C2DED7                 JNZ   :D7DE     If file type byte <>0:
502                                             Run error
503 D788 F1                     POP   PSW
504 D789 C9                     RET
505                     *
506                     ****************************************
507                     * CHECK IF LOAD DURING RUN PROGRAM *
508                     ****************************************
509                     *
510                     * Entry: C: 00 if load during run, else it is FF.
511                     *         A: File type byte.
512                     * Exit:  ABCDEHL preserved.
513                     *
514 D78A 0D             MPT24   DCR   C
515 D78B 0C                     INR   C         Check C
516 D78C C8                     RZ              Abort if during run
517 D78D C3EBD7                 JMP   :D7EB     Display file type byte
518                     *
519                     *
520                     *
521 D790                        END
```

```
****************************
* S Y M B O L   T A B L E *
****************************
```

ASKKEY	D6A5	BREAK	D6A5	FGETC	D6BB	GETC	D6BE
GTC10	D6D4	KFSCAN	D620	KPTRU	D69C	LD67	D64E
MPR29	D6C1	MPT00	D72D	MPT11	D71A	MPT13	D668
MPT14	D678	MPT15	D687	MPT18	D681	MPT20	D6E5
MPT21	D720	MPT24	D78A	MPT28	D750	MPT30	D743
MPT31	D695	MPT38	D65F	MPT39	D658	MPT49	D73D
MPT54	D783	OTC30	D74C	OTK10	D6B9	RLA15	D6ED
RMI15	D658	SINKEY	D63F	SMA20	D70D	SMA30	D70F
SNTMP	D755	SPT00	D68D	SPT01	D6F5	SPTA2	D706
SSM20	D700	TKEY	D632	TKY10	D635	TOUTSE	D642
WSPACE	D6DA						

```
002                             ORG   :D790
003                     *
004                     *
005                     *
006                     ************************
007                     * CHECK FREE RAM SPACE *
008                     ************************
009                     *
010                     * Entry: DE: Startaddress.
011                     *        HL: 1st not useable address.
012                     * Exit:  HL: Useable RAM space.
013                     *        ABCDE preserved.
014                     *
015 D790 CD1ADE         MPT25   CALL  :DE1A     Calculate free space
016 D793 2B                     DCX   H
017 D794 C9                     RET
018                     *
019                     ************************************
020                     * TRANSFER DATA/CASSETTE VECTORS *
021                     ************************************
022                     *
023                     * Transfer data/cassette switching vectors from
024                     * ROM to RAM vector area.
025                     *
026                     * Exit: AFBC preserved. DEHL corrupted.
027                     *
028                     MPT01
029 D795 C5             CASIN   PUSH  B
030 D796 21CBD7                 LXI   H,:D7CB   Highest source address
031 D799 11A4D7                 LXI   D,:D7A4   Lowest source address
032 D79C 01C502                 LXI   B,:02C5   Lowest destination address
033 D79F CD4FDE                 CALL  :DE4F     Transfer cassette vectors
034 D7A2 C1                     POP   B
035 D7A3 C9                     RET
036                     *
037                     ************************************
038                     * DATA/CASSETTE SWITCHING VECTORS *
039                     ************************************
040                     *
041                     * This data block is moved into the RAM area
042                     * #02C5-#02EB during system initialisation.
043                     *
044 D7A4 C3B8D2         CINTB   JMP   :D2B8     WOPEN
045 D7A7 C3F1D2                 JMP   :D2F1     WBLK
046 D7AA C327D4                 JMP   :D427     WCLOSE
047 D7AD C325D3                 JMP   :D325     ROPEN
048 D7B0 C340D3                 JMP   :D340     RBLK
049 D7B3 C345D4                 JMP   :D445     RCLOSE
050 D7B6 C3A2D3                 JMP   :D3A2     MBLK
051 D7B9 C9                     RET             RESET
052 D7BA 00                     NOP
053 D7BB 00                     NOP
054 D7BC C9                     RET             DOUTC
055 D7BD 00                     NOP
056 D7BE 00                     NOP
057 D7BF C3B4DD                 JMP   :DDB4     DINC
058 D7C2 C9                     RET
059 D7C3 00                     NOP
060 D7C4 00                     NOP
061 D7C5 2424                   DATA  :24,:24   TAPSL
062 D7C7 243C                   DATA  :24,:3C   TAPSD
063 D7C9 2418                   DATA  :24,:18   TAPST
```

```
064                     *
065                     ************************************
066                     * WAIT FOR SPACEBAR, PRINT CAR.RET *
067                     ************************************
068                     *
069                     * Exit: BCDEHL preserved.
070                     *       CY=1: Break pressed.
071                     *
072                     CINTE
073 D7CB CDDAD6        WPT     CALL  :D6DA     Wait for spacebar
074 D7CE DA0CC8                 JC    :C80C     If BREAK pressed: into
075                                             Basic monitor
076 D7D1 C35EDD                JMP   :DD5E     Print car.ret; ret
077                     *
078 D7D4 FF                    DATA  :FF
079 D7D5 FF                    DATA  :FF
080 D7D6 FF                    DATA  :FF
081 D7D7 FF                    DATA  :FF
082                     *
083                     ***********************************
084                     * WRITE BLOCK + TRAILER ON TAPE *
085                     ***********************************
086                     *
087                     * Entry: DE: Length block.
088                     *        HL: Startaddress block.
089                     * Exit:  A=0, BCDE preserved.
090                     *        HL points past block written.
091                     *
092 D7D8 CDC802        MPT10   CALL  :02C8     Write block
093 D7DB C3CB02                JMP   :02CB     Write trailer
094                     *
095                     *************************
096                     * FAILURE DURING ROPEN *
097                     *************************
098                     *
099 D7DE 0D            RPN20   DCR   C
100 D7DF 0C                    INR   C         Load during program ?
101 D7E0 C45EDD                CNZ   :DD5E     Print car.ret if not
102 D7E3 C329D3                JMP   :D329     Back to read file leader
103                     *
104                     ****************************
105                     * CHECK FILE OF ANY TYPE *
106                     ****************************
107                     *
108                     MPT12
109 D7E6 3E00          AMBLK   MVI   A,:00     File type correct
110 D7E8 C3D702                JMP   :02D7     Read and check program name
111                     *
112                     ******************************************
113                     * WRITE BYTE ON CURSOR POSITION ADDRESS *
114                     * AND UPDATE CURSOR POSITION.           *
115                     ******************************************
116                     *
117                     * This routine is a fast printing routine:
118                     * the data byte is poked directly into the
119                     * screen RAM.
120                     *
121                     * Entry: Byte to be written on screen in A.
122                     * Exit:  All registers preserved.
123                     *
124 D7EB E5            LD95    PUSH  H
125 D7EC 00                    NOP
```

```
126 D7ED 2A7200             LHLD  :0072     Get cursor position address
127 D7F0 77                 MOV   M,A       Write byte on screen
128 D7F1 2B                 DCX   H         ) Update cursor addr
129 D7F2 2B                 DCX   H         )
130 D7F3 227200             SHLD  :0072     Save new cursor address
131 D7F6 E1                 POP   H
132 D7F7 C9                 RET
133                 *
134                 ****************************
135                 * SAVE: WRITE NAME LENGTH *
136                 ****************************
137                 *
138                 * Entry: HL: Addr. length byte of name.
139                 * Exit:  DE: Length of name.
140                 *        HL: Points past string.
141                 *        BC: Preserved.
142                 *        A:  Checksum on string.
143                 *
144 D7F8 5E          MPT22   MOV   E,M       Get name length
145 D7F9 1600               MVI   D,:00
146 D7FB 23                 INX   H         HL to 1st byte of name
147 D7FC C3F1D2             JMP   :D2F1     Write name length
148                 *
149                 ********************************
150                 * INITIALISE LOADING FROM TAPE *
151                 ********************************
152                 *
153                 * Entry: HL: Points to length byte of name requested
154                 * Exit:  DE: Length requested name.
155                 *        HL: Points to first byte of name.
156                 *        AFBC preserved.
157                 *
158 D7FF 5E          MPT23   MOV   E,M       Get length requested name
159 D800 1600               MVI   D,:00     in DE
160 D802 23                 INX   H         HL points to 1st byte name
161 D803 C32ED4             JMP   :D42E     Cassette motors on; ret
162                 *
163                 ***********************
164                 * UPDATE POROM/PORO *
165                 ***********************
166                 *
167                 * Entry: MPT52: Byte for ROM/cassette select in A.
168                 *        MPT53: New POROM byte.
169                 *
170 D806 E6F0        MPT52   ANI   :F0       Enable ROM/cassette select
171 D808 324000      MPT53   STA   :0040     Load POROM
172 D80B 3206FD             STA   :FD06     and PORO
173 D80E C9                 RET
174                 *
175                 ******************
176                 * part of 2EAC1 *
177                 ******************
178                 *
179                 * If pointer is off top visible screen:
180                 *
181 D80F 7A          PCK40   MOV   A,D
182 D810 FED0               CPI   :D0
183 D812 DAEFEA             JC    :EAEF     (2) if <= CF (no overflow
184                                          below 0)
185 D815 E5                 PUSH  H         ) if > CF:
186 D816 C5                 PUSH  B         ) Exchange BC and HL
187 D817 E1                 POP   H         )
```

```
188 D818 C1                     POP   B          )
189 D819 C3EFEA                 JMP   :EAEF      (2)
190                     *
191 D81C FF                     DATA  :FF
192                     *
193                     ************************
194                     * RUN basiccmd SAVEA *
195                     ************************
196                     *
197 D81D CD3BD8         RSAVA   CALL  :D83B      Evaluate array type to be
198                                              saved
199 D820 F5                     PUSH  PSW        Save type array
200 D821 C226D8                 JNZ   :D826      Jump if INT/FPT array
201 D824 E7                     RST   4          Move stringarray into one
202 D825 75                     DATA  :75        string in free RAM
203 D826 F1             LD99    POP   PSW        Get type array
204 D827 E5                     PUSH  H
205 D828 D5                     PUSH  D
206 D829 F5                     PUSH  PSW
207 D82A CD91E7                 CALL  :E791      (0) Evaluate program name
208 D82D 3E32                   MVI   A,:32      File type byte '2'
209 D82F CDC502                 CALL  :02C5      WOPEN
210 D832 F1                     POP   PSW        Get type array
211 D833 213E01                 LXI   H,:013E    Startaddr EBUF
212 D836 77                     MOV   M,A        Type into EBUF
213 D837 C368D6                 JMP   :D668      Write 2 blocks + trailer
214                     *
215 D83A 00                     NOP
216                     *
217                     ********************************************
218                     * EVALUATE ARRAY TYPE TO BE SAVED/LOADED *
219                     ********************************************
220                     *
221                     * Exit: A:    Array type.
222                     *       DE:   Length all array elements.
223                     *       HL:   Beginaddr. 1st array element.
224                     *       Z=1:  String array.
225                     *       Z=0:  INT/FPT array.
226                     *
227 D83B CD5AE9          RLSAS   CALL  :E95A      (0) Get array addr in HL
228 D83E E630                   ANI   :30        Allow any type array
229 D840 F5                     PUSH  PSW        Save type
230 D841 5E                     MOV   E,M        )
231 D842 23                     INX   H          ) Get array pntr in DE
232 D843 56                     MOV   D,M        )
233 D844 7B                     MOV   A,E
234 D845 B2                     ORA   D
235 D846 CA90E9                 JZ    :E990      (0) Undef. array if no addr
236 D849 EB                     XCHG             Pointer to array in HL
237 D84A 2B                     DCX   H
238 D84B 2B                     DCX   H
239 D84C 56                     MOV   D,M        Get 1st length byte
240 D84D 23                     INX   H
241 D84E 7E                     MOV   A,M        Get 2nd length byte
242 D84F 23                     INX   H
243 D850 96                     SUB   M          Minus nr of dim. bytes
244 D851 5F                     MOV   E,A
245 D852 7A                     MOV   A,D
246 D853 DE00                   SBI   :00        Update hibyte if borrow
247 D855 57                     MOV   D,A
248 D856 1B                     DCX   D          DE is length all array elem.
249 D857 CD39DE                 CALL  :DE39      HL is beginaddr 1st element
```

```
D85A F1                 POP   PSW       Get type in A
D85B FE20               CPI   :20       Set Z-flag on type
D85D C9                 RET
                *
                ************************
                * RUN basiccmd LOADA *
                ************************
                *
D85E CD3BD8     RLODA   CALL  :D83B      Evaluate array type to be
                                         loaded
D861 E5                 PUSH  H          Save startaddr array elem.
D862 F5                 PUSH  PSW        Save type
D863 CD78D6             CALL  :D678      Evaluate requested program
                                         name; prep. select ROM bank
D866 CD08D8             CALL  :D808      Select ROM bank 1
D869 F1                 POP   PSW        Get type
D86A C30FEE             JMP   :EE0F      (1) Read block from tape and
                                         store it in array
                *
                ************************************
                * INITIALISE 'EDIT': EMPTY HEAP,   *
                * CLEAR SYMBOL TABLE, MOVE PROGRAM *
                ************************************
                *
                * Part of 'Init. EDIT' (0E265).
                *
                * Entry: CY=1: Not sufficient memory available.
                *
D86D DA10DA     MPT33   JC    :DA10      Evt. run error 'OUT OF
                                         MEMORY'
D870 C323CB             JMP   :CB23      Empty heap, move
                                         program, clear symtab
                *
                ********************************************
                * LIST CURRENT LINE IF LINENR IS CORRECT *
                ********************************************
                *
                * Part of 'run LIST <range>' (0E1B6).
                *
                * Entry: CY=0 if linenr. <= 0 or > FFFF.
                *
D873 D215DA     MPT3A   JNC   :DA15      Evt. run error 'NUMBER
                                         OUT OF RANGE'
D876 C3ABEC             JMP   :ECAB      (0) List current line
                *
                ***************************
                * INPUT FROM EDIT BUFFER *
                ***************************
                *
                * Part of 'restart interpreter' (C823).
                *
D879 E5         MPT05   PUSH  H
D87A CD91E2             CALL  :E291      (0) Input from editbuffer
D87D E1                 POP   H
D87E C9                 RET
                *
                ********************************
                * part of RUN 'CLEAR' (0E6B5) *
                ********************************
                *
                * Checks for heap too big.
                *
```

```
312 D87F CDF8E6     MPT42   CALL  :E6F8     (0) Get space req. in HL
313 D882 7C                 MOV   A,H
314 D883 B7                 ORA   A         Set flags on hibyte
315 D884 37                 STC             CY=1 if > 32K
316 D885 C9                 RET
317                 *
318                 ************************************
319                 * EVT. INITIALISE 4 COLOUR ANIMATE *
320                 ************************************
321                 *
322                 * Part of 2E9C3.
323                 *
324 D886 FE14       SPT04   CPI   :14
325 D888 D0                 RNC             Abort if A >= 14
326 D889 32C100             STA   :00C1     Set for 4-colour animate
327 D88C C9                 RET
328                 *
329                 ********
330                 * DATA *
331                 ********
332                 *
333 D88D 20         LD221   DATA  :20       Space
334 D88E 8C72               DATA  :8C,:72   Pntr. to 'MODE'
335 D890 00                 DATA  :00
336                 *
337                 ********************************
338                 * part of 1EE0B - (not used) *
339                 ********************************
340                 *
341 D891 F5         LD105   PUSH  PSW
342 D892 CD91E7             CALL  :E791     (1)
343 D895 F1                 POP   PSW
344 D896 C9                 RET
345                 *
346                 ***********************************************
347                 * READ BLOCK FROM TAPE, EVT. ERROR REPORT *
348                 ***********************************************
349                 *
350                 * Part of LOADA (1EE0F).
351                 *
352 D897 CDD102       MPT19   CALL  :02D1     Read block from tape
353 D89A D2B3D2               JNC   :D2B3     Evt. run 'LOADING ERROR'
354 D89D C9                   RET
355                 *
356                 ************************************
357                 * LIST ARRAY NAME, SPACE, EXPRESSION *
358                 ************************************
359                 *
360                 * Used in listing 'Savea/Loada' textlines.
361                 *
362 D89E CDF7EE       SCN30   CALL  :EEF7     (0) List array name
363 D8A1 C365ED               JMP   :ED65     (0) List space, expression
364                 *
365 D8A4 FF                   DATA  :FF
366 D8A5 FF                   DATA  :FF
367                 *
368                 ********************************
369                 * INITIALISE SOUND GENERATOR *
370                 ********************************
371                 *
372                 * All sound channels are switched off.
373                 *
```

```
                        * Exit: AB preserved, CDEHLF corrupted.
                        *
D8A6 2106FC     SNDINI  LXI   H,:FC06   )
D8A9 3636               MVI   M,:36     ) Load 3 timers
D8AB 3676               MVI   M,:76     )
D8AD 36B6               MVI   M,:B6     )
D8AF 210000             LXI   H,:0000
D8B2 2204FD             SHLD  :FD04     Volume 4 channels off
D8B5 229402             SHLD  :0294     and remember it
D8B8 21C201             LXI   H,:01C2   Start 1st SCB
D8BB 110E00             LXI   D,:000E   Length SCB
D8BE 0E04               MVI   C,:04     4 blocks (3 SCB, 1 NCB)
D8C0 36FF       SNI10   MVI   M,:FF     FF in 1st byte SCB (= off)
D8C2 19                 DAD   D         Calc start next block
D8C3 0D                 DCR   C
D8C4 C2C0D8             JNZ   :D8C0     Next block if not ready
D8C7 C9                 RET
                        *
                        **********************
                        * OUTPUT TO DCE-BUS *
                        **********************
                        *
                        * 'Real World' output. Writes a byte to a given
                        * Real World address.
                        *
                        * Entry: D: Busaddress.
                        *        E: Data for output.
                        *
D8C8 F5         RWOP    PUSH  PSW
D8C9 E5                 PUSH  H
D8CA 2103FE             LXI   H,:FE03   GIC control address
D8CD 3680               MVI   M,:80     All ports output
D8CF 2B                 DCX   H         Port C addr. in HL
D8D0 36FE               MVI   M,:FE     Clear bus expand signal
D8D2 EB                 XCHG            Data in L, busaddr. in H
D8D3 2200FE             SHLD  :FE00     Data in PA, busaddr. in PB
D8D6 EB                 XCHG            Address PC in HL
D8D7 34                 INR   M         Set bus expand signal
D8D8 36FD               MVI   M,:FD     Set write strobe true
                                        (Now data exchange done)
D8DA 36FF               MVI   M,:FF     Reset strobe
D8DC 35                 DCR   M         Clear bus expand signal
D8DD E1                 POP   H
D8DE F1                 POP   PSW
D8DF C9                 RET
                        *
                        **********************
                        * INPUT FROM DCE-BUS *
                        **********************
                        *
                        * 'Real World' input. Reads a byte from a given
                        * Real World address.
                        *
                        * Entry: D: Busaddress.
                        * Exit:  E: Data received.
                        *
D8E0 F5         RWIP    PUSH  PSW
D8E1 E5                 PUSH  H
D8E2 2103FE             LXI   H,:FE03   GIC control addr. in HL
D8E5 3690               MVI   M,:90     PA input, rest output
D8E7 2B                 DCX   H         Address PC in HL
D8E8 36FE               MVI   M,:FE     Clear bus expand signal
```

```
436 D8EA 7A                    MOV   A,D       Busaddress in A
437 D8EB 3201FE                STA   :FE01     Store busaddress in PB
438 D8EE 34                    INR   M         Set bus expand signal
439 D8EF 36FB                  MVI   M,:FB     Set read strobe true
440                                            (Now data exchange)
441 D8F1 3A00FE                LDA   :FE00     Data to A
442 D8F4 5F                    MOV   E,A       Data in E
443 D8F5 36FF                  MVI   M,:FF     Reset strobe
444 D8F7 35                    DCR   M
445 D8F8 E1                    POP   H
446 D8F9 F1                    POP   PSW
447 D8FA C9                    RET
448                  *
449                  *
450                  *
451 D8FB                       END
```

```
****************************
* S Y M B O L   T A B L E *
****************************
```

AMBLK	D7E6	CASIN	D795	CINTB	D7A4	CINTE	D7CB
LD105	D891	LD221	D88D	LD95	D7EB	LD99	D826
MPT01	D795	MPT05	D879	MPT10	D7D8	MPT12	D7E6
MPT19	D897	MPT22	D7F8	MPT23	D7FF	MPT25	D790
MPT33	D86D	MPT3A	D873	MPT42	D87F	MPT52	D806
MPT53	D808	PCK40	D80F	RLODA	D85E	RLSAS	D83B
RPN20	D7DE	RSAVA	D81D	RWIP	D8E0	RWOP	D8C8
SCN30	D89E	SNDINI	D8A6	SNI10	D8C0	SPT04	D886
WPT	D7CB						

```
002                             ORG   :D8FB
003                     *
004                     *
005                     *
006                     *  ==================
007                     *** INTERRUPT HANDLER ***
008                     *  ==================
009                     *
010                     *
011                     ********************************
012                     * INITIALISE INTERRUPT SYSTEM *
013                     ********************************
014                     *
015                     * Initialises the interrupt system of the machine.
016                     *
017                     * Entry: No conditions.
018                     * Exit:  All registers corrupted.
019                     *
020 D8FB 21F8FF         INTINI  LXI   H,:FFF8   Address int.mask register
021 D8FE 3E04                   MVI   A,:04
022 D900 77                     MOV   M,A       Only stack interrupt
023 D901 325F00                 STA   :005F     Remember int. mask
024 D904 2EF4                   MVI   L,:F4
025 D906 3E0C                   MVI   A,:0C
026 D908 77                     MOV   M,A       Select ext.int. and INTA
027 D909 3C                     INR   A
028 D90A 77                     MOV   M,A       Reset
029 D90B 3E0C                   MVI   A,:0C
030 D90D 32C001                 STA   :01C0     Init. cursor timer
031 D910 3E02                   MVI   A,:02
032 D912 32C101                 STA   :01C1     Init. keyboard scan counter
033 D915 CDA3D9                 CALL  :D9A3     Reload sound timer
034 D918 CD9DD9                 CALL  :D99D     Reload keyboard timer
035 D91B CD49D9                 CALL  :D949     Set up int. entry points
036
037                     * Set-up interrupt vectors (also entry from utility)
038
039 D91E 21A9D9         INTSU   LXI   H,:D9A9
040 D921 227000                 SHLD  :0070     Clock int. vector (7)
041 D924 2178D5                 LXI   H,:D578
042 D927 226E00                 SHLD  :006E     Keyboard int. vector (6)
043 D92A 21FDC6                 LXI   H,:C6FD
044 D92D 226C00                 SHLD  :006C     Screen restart vector (5)
045 D930 21C0C6                 LXI   H,:C6C0
046 D933 226A00                 SHLD  :006A     Math. restart vector (4)
047 D936 2155D7                 LXI   H,:D755
048 D939 226800                 SHLD  :0068     Sound int. vector (3)
049 D93C 21E2D9                 LXI   H,:D9E2
050 D93F 226600                 SHLD  :0066     Stack int. vector (2)
051 D942 210EC7                 LXI   H,:C70E
052 D945 226400                 SHLD  :0064     Utility/encode vector (1)
053 D948 C9                     RET
054                     *
055                     * SET UP INTERRUPT VECTOR AREA:
056                     *
057                     * Sets up the vector area 0000-003F.
058                     *
059                     * Entry: No conditions.
060                     * Exit:  All registers corrupted.
061                     *
062 D949 010000         VECSU   LXI   B,:0000   Start at zero
063 D94C 116BD9         VCS10   LXI   D,:D96B   Startaddr. int. routine
```

```
064 D94F 2173D9             LXI   H,:D973   Endaddress
065 D952 CD4FDE             CALL  :DE4F     Transfer int. routine
066 D955 79                 MOV   A,C
067 D956 FE40               CPI   :40       8 routines done?
068 D958 C24CD9             JNZ   :D94C     Next one if not ready
069 D95B 213B00             LXI   H,:003B   Addr. highest int.vector
070 D95E 01F8FF             LXI   B,:FFF8
071 D961 1670               MVI   D,:70
072 D963 72          VCS20  MOV   M,D       Load vector addr.
073 D964 15                 DCR   D
074 D965 15                 DCR   D
075 D966 09                 DAD   B         Calculate next addr
076 D967 DA63D9             JC    :D963     Next one if not ready
077 D96A C9                 RET
078                  *
079                  * INTERRUPT VECTOR ROUTINE:
080                  *
081                  * During initialisation loaded into RAM on the
082                  * Restart routine locations of the CPU.
083                  * The address after LHLD is later changed into
084                  * the appropriate vector address by #D8FB.
085                  *
086 D96B 00           ITMPL  NOP
087 D96C E5                 PUSH  H
088 D96D 2A7000             LHLD  :0070
089 D970 E9                 PCHL
090 D971 00                 NOP
091 D972 00                 NOP
092                  *
093                  ********************************
094                  * DISABLE KEYBOARD INTERRUPTS *
095                  ********************************
096                  *
097                  * Disables keyboard interrupts only.
098                  *
099                  * Entry can be different. From INTCH common part
100                  * to disable/enable sound, clock and keyboard
101                  * interrupts.
102                  *
103                  * Entry: None.
104                  * Exit:  All registers preserved.
105                  *
106 D973 C5          KBDI   PUSH  B
107 D974 0100BF             LXI   B,:BF00   Disable keyboard interrupts
108                  *
109 D977 F5          INTCH  PUSH  PSW
110 D978 F3                 DI              Disable interrupts
111 D979 3A5F00             LDA   :005F     Get current int.mask
112 D97C A0                 ANA   B         ) Calculate new one
113 D97D B1                 ORA   C         )
114 D97E 325F00             STA   :005F     Remember new mask
115 D981 32F8FF             STA   :FFF8     and set mask
116 D984 FB                 EI              Enable interrupts again
117 D985 F1                 POP   PSW
118 D986 C1                 POP   B
119 D987 C9                 RET
120                  *
121                  ********************************
122                  * ENABLE KEYBOARD INTERRUPTS *
123                  ********************************
124                  *
```

```
126                     *
127 D988 C5             KBEI    PUSH  B
128 D989 0140FF                 LXI   B,:FF40   Enable keyboard interrupts
129 D98C C377D9                 JMP   :D977     Update int. mask
130                     *
131                     ******************************
132                     * DISABLE SOUND INTERRUPTS *
133                     ******************************
134                     *
135                     * Disables sound interrupts only.
136                     *
137 D98F C5             SNDDI   PUSH  B
138 D990 0100F7                 LXI   B,:F700   Disable sound interrupts
139 D993 C377D9                 JMP   :D977     Update int. mask
140                     *
141                     ****************************
142                     * ENABLE SOUND INTERRUPTS *
143                     ****************************
144                     *
145                     * Enables sound interrupts only.
146                     *
147 D996 C5             SNDEI   PUSH  B
148 D997 0108FF                 LXI   B,:FF08   Enable sound interrupts
149 D99A C377D9                 JMP   :D977     Update int. mask
150                     *
151                     ************************
152                     * LOAD KEYBOARD TIMER *
153                     ************************
154                     *
155                     * Starts a keyboard interrupt.
156                     *
157 D99D 3EFF           KBIS    MVI   A,:FF     Init. 16.32 msec
158 D99F 32FCFF                 STA   :FFFC     Load timer 4 (keyboard)
159 D9A2 C9                     RET
160                     *
161                     *********************
162                     * LOAD SOUND TIMER *
163                     *********************
164                     *
165                     * Starts a sound interrupt.
166                     *
167 D9A3 3E50           SNDIS   MVI   A,:50     Init. 5.12 msec
168 D9A5 32FBFF                 STA   :FFFB     Load timer 3 (sound)
169 D9A8 C9                     RET
170                     *
171                     ****************************
172                     * CLOCK INTERRUPT (RST 7) *
173                     ****************************
174                     *
175                     * Triggered every 20 msec by the TV page
176                     * blanking signal.
177                     * Decrements timer 01BE/BF until 0. Checks also
178                     * the contents of cursor clock timer 01C0. It is
179                     * decremented; if it becomes 0, the timer is re-
180                     * loaded and the cursor is flashed.
181                     *
182                     * Exit: All registers preserved.
183                     *
184 D9A9 F5             CLKINT  PUSH  PSW
185 D9AA C5                     PUSH  B
186 D9AB D5                     PUSH  D
187 D9AC 3A5F00                 LDA   :005F     Get current int. mask
```

```
188 D9AF F5                 PUSH  PSW       and remember it
189 D9B0 3E04               MVI   A,:04
190 D9B2 32F8FF             STA   :FFF8     Set stack interrupt only
191 D9B5 FB                 EI
192 D9B6 2ABE01             LHLD  :01BE     Get timer contents
193 D9B9 7C                 MOV   A,H
194 D9BA B5                 ORA   L
195 D9BB CAC2D9             JZ    :D9C2     If timer = 0
196 D9BE 2B                 DCX   H         Else:
197 D9BF 22BE01             SHLD  :01BE     decrement timer
198 D9C2 21C001     CKI10   LXI   H,:01C0   Addr. cursor clock
199 D9C5 35                 DCR   M         Decr. clock
200 D9C6 C2CDD9             JNZ   :D9CD     Return if <>0
201 D9C9 360F               MVI   M,:0F     Load 20 ms flash time
202 D9CB EF                 RST   5         Flash cursor
203 D9CC 12                 DATA  :12
204                 *
205                 * GENERAL INTERRUPT RETURN:
206                 *
207                 * Restores interrupt mask and all registers.
208                 *
209                 * Entry: Interrupt mask and all registers on stack.
210                 *
211 D9CD F1           INTRM   POP   PSW       Get old int. mask
212 D9CE F3                   DI
213 D9CF 32F8FF             STA   :FFF8     Restore int. mask
214 D9D2 325F00             STA   :005F     and remember it.
215 D9D5 FB                 EI
216 D9D6 D1                 POP   D
217 D9D7 C1                 POP   B
218 D9D8 F1                 POP   PSW
219 D9D9 E1                 POP   H
220 D9DA C9                 RET
221                 *
222                 ****************************
223                 * ENABLE CLOCK INTERRUPT *
224                 ****************************
225                 *
226 D9DB C5           CLKEI   PUSH  B
227 D9DC 0180FF             LXI   B,:FF80   Enable clock interrupt
228 D9DF C377D9             JMP   :D977     Update int.mask
229                 *
230                 ****************************
231                 * STACK INTERRUPT (RST2) *
232                 ****************************
233                 *
234                 * This routine handles an interrupt caused by the
235                 * stack overflow hardware logic.
236                 *
237 D9E2 3100F9       SPINT   LXI   SP,:F900  Reset stackpointer
238 D9E5 AF                   XRA   A
239 D9E6 321701             STA   :0117     No running inputs
240 D9E9 322201             STA   :0122     No encoding of stored line
241 D9EC CD43D7             CALL  :D743     Input from keyboard, reload
242                                          timers sound/keyb
243 D9EF FB                 EI
244 D9F0 3E16               MVI   A,:16
245 D9F2 C3F5D9             JMP   :D9F5     Run error 'STACK OVERFLOW'
246                 *
247                 *
248                 *
249                 *
```

```
                        *  ==============
                        *** ERROR HANDLER ***
                        *  ==============
                        *
                        *
                        ******************
                        * ERROR HANDLING *
                        ******************
                        *
                        * Produces an error message and other information
                        * about errors.
                        *
                        * Entry: A:  Error message number.
                        *        BC: Latest position in EBUF or textbuffer.
                        * Exit:  If during input: Restart input statement.
                        *        If during encoding: Back to ELINA (C93C)
                        *                            for handling.
                        *        Else: Restart Basic (direct mode).
                        *
D9F5 47         ERROR   MOV   B,A       Save error pointer
D9F6 AF                 XRA   A
D9F7 323101             STA   :0131     Set output screen/RS232
D9FA CDE4CE             CALL  :CEE4     Select ROM bank 0
D9FD 0000               DBL   :0000
D9FF 00                 NOP
DA00 00                 NOP
DA01 3A1801             LDA   :0118     Get RUNF
DA04 B7                 ORA   A
DA05 C23DDA             JNZ   :DA3D     If run-time error
DA08 C364DA             JMP   :DA64     If compile-time error
                        *
                        * ENTRYPOINTS TO ERROR ROUTINES:
                        *
DA0B 3E17       ERRSN   MVI   A,:17
DA0D C3F5D9             JMP   :D9F5     Run 'SYNTAX ERROR'
                        *
DA10 3E13       ERROM   MVI   A,:13
DA12 C3F5D9             JMP   :D9F5     Run 'OUT OF MEMORY'
                        *
DA15 3E09       ERRRA   MVI   A,:09
DA17 C3F5D9             JMP   :D9F5     Run 'NUMBER OUT OF RANGE'
                        *
DA1A 3E14       ERRTM   MVI   A,:14
DA1C C3F5D9             JMP   :D9F5     Run 'TYPE MISMATCH'
                        *
DA1F 3E03       ERROV   MVI   A,:03
DA21 C3F5D9             JMP   :D9F5     Run 'OVERFLOW'
                        *
DA24 3E06       ERRDO   MVI   A,:06
DA26 C3F5D9             JMP   :D9F5     Run 'DIVISION BY ZERO'
                        *
DA29 3E05       ERRBS   MVI   A,:05
DA2B C3F5D9             JMP   :D9F5     Run 'SUBSCRIPT ERROR'
                        *
DA2E 3E00       ERRNF   MVI   A,:00
DA30 C3F5D9             JMP   :D9F5     Run 'NEXT WITHOUT FOR'
                        *
DA33 3E12       ERREL   MVI   A,:12
DA35 C3F5D9             JMP   :D9F5     Run 'ERROR LINE RUN'
                        *
DA38 3E08       ERRLS   MVI   A,:08
DA3A C3F5D9             JMP   :D9F5     Run 'STRING TOO LONG'
```

```
                        *
                        ******************
                        * RUN-TIME ERROR *
                        ******************
                        *
DA3D CD50DA     ERRRU   CALL  :DA50     Print error message
DA40 3A1701             LDA   :0117     Get RDIPL
DA43 B7                 ORA   A
DA44 C24DDA             JNZ   :DA4D     Jump if running inputs
DA47 CD75DA             CALL  :DA75     else: Print 'IN LINE <nr>'
DA4A C314C8             JMP   :C814     Re-enter BASIC

                        * If error in input:

DA4D C350E3     ERT10   JMP   :E350     (0) Back to restart input
                        *
                        ************************
                        * PRINT ERROR MESSAGE *
                        ************************
                        *
                        * Entry: B: Error message number.
                        * Exit:  BC preserved, AFDEHL corrupted.
                        *
DA50 CD55DD     ERRMS   CALL  :DD55     Cursor to begin (next) line
DA53 78                 MOV   A,B       Get error number
DA54 2194DA             LXI   H,:DA94   Base of list error messages
DA57 87                 ADD   A         Multiply error nr *2
DA58 5F                 MOV   E,A       and save offset
DA59 1600               MVI   D,:00
DA5B 19                 DAD   D         Calculate table addr.
DA5C 5E                 MOV   E,M       ) Get addr of message
DA5D 23                 INX   H         ) in DE
DA5E 56                 MOV   D,M       )
DA5F EB                 XCHG            and in HL
DA60 CDD4DA             CALL  :DAD4     Print error message
DA63 C9                 RET
                        *
                        ***********************
                        * COMPILE-TIME ERROR *
                        ***********************
                        *
                        * Handles errors in direct mode. Only an error
                        * message is printed. Control is passed back to
                        * ELINA (C93C) except if it is an error during
                        * encoding of a stored line.
                        *
                        * Entry: A: Error number.
                        *        C: Points to last read character on
                        *           input line.
                        * Exit:  To Basic interpreter.
                        *
DA64 3A2201     ERRCO   LDA   :0122     Get ERSFL
DA67 FE01               CPI   :01       Error during encoding?
DA69 CA57C9             JZ    :C957     Then handle error

                        * Error in direct command:

                        CALL  :DA50     Print error message
DA6C CD50DA             CALL  :DD5E     Print car.ret
DA6F CD5EDD             JMP   :C823     Restart interpreter
DA72 C323C8             *
```

```
374                     ********************************************
375                     * PRINT LINE NUMBER IN WHICH ERROR OCCURED *
376                     ********************************************
377                     *
378                     * Prints car.ret if current line is a direct comman
379                     * Else, prints 'IN LINE <linenr>' and car.ret.
380                     *
381                     * Entry: None.
382                     * Exit:  ABCDEHL preserved. F corrupted.
383                     *        Z=1: direct command.
384                     *
385 DA75 E5           MSGIL   PUSH  H
386 DA76 F5                   PUSH  PSW
387 DA77 2A0001             LHLD  :0100     Get start current line
388 DA7A 7C                 MOV   A,H
389 DA7B B5                 ORA   L         Check if 0
390 DA7C F5                 PUSH  PSW
391 DA7D CA8CDA             JZ    :DA8C     If direct, print car.ret
392 DA80 CDFFDA             CALL  :DAFF     Else, print 'IN LINE'
393 DA83 7FDB               DBL   :DB7F
394 DA85 7E                 MOV   A,M       ) Get line nr in HL
395 DA86 23                 INX   H         )
396 DA87 6E                 MOV   L,M       )
397 DA88 67                 MOV   H,A       )
398 DA89 CDB4EF             CALL  :EFB4     (0) Print line number
399 DA8C CD5EDD         MIL10   CALL  :DD5E     Print car.ret
400 DA8F F1                 POP   PSW
401 DA90 E1                 POP   H
402 DA91 7C                 MOV   A,H
403 DA92 E1                 POP   H
404 DA93 C9                 RET
405                     *
406                     ************************************
407                     * ERROR MESSAGES INDIRECTION TABLE *
408                     ************************************
409                     *
410                     * The address points to the location where the
411                     * string with the error messages can be found.
412                     * Between brackets the error number.
413                     *
414                     * Run-time errors:
415                     ERITB
416 DA94 1CDC          E?NF    DBL   :DC1C     (00) NEXT WITHOUT FOR
417 DA96 2CDC          E?RG    DBL   :DC2C     (01) RETURN WITHOUT GOSUB
418 DA98 33DC          E?OD    DBL   :DC33     (02) OUT OF DATA
419 DA9A 3EDC          E?OV    DBL   :DC3E     (03) OVERFLOW
420 DA9C 50DC          E?US    DBL   :DC50     (04) UNDEFINED LINE NUMBER
421 DA9E 5CDC          E?BS    DBL   :DC5C     (05) SUBSCRIPT ERROR
422 DAA0 68DC          E?DO    DBL   :DC68     (06) DIVISION BY ZERO
423 DAA2 95DC          E?OS    DBL   :DC95     (07) OUT OF STRING SPACE
424 DAA4 9DDC          E?LS    DBL   :DC9D     (08) STRING TOO LONG
425 DAA6 D8DC          E?RA    DBL   :DCD8     (09) NUMBER OUT OF RANGE
426 DAA8 B2DC          E?IN    DBL   :DCB2     (0A) INVALID NUMBER
427 DAAA FADC          E?LO0   DBL   :DCFA     (0B) LOADING ERROR 0
428 DAAC FEDC          E?LO1   DBL   :DCFE     (0C) LOADING ERROR 1
429 DAAE 02DD          E?LO2   DBL   :DD02     (0D) LOADING ERROR 2
430 DAB0 06DD          E?LO3   DBL   :DD06     (0E) LOADING ERROR 3
431 DAB2 F1DC          E?UA    DBL   :DCF1     (0F) UNDEFINED ARRAY
432 DAB4 C2DC          E?NC    DBL   :DCC2     (10) COLOUR NOT AVAILABLE
433 DAB6 B7DC          E?OF    DBL   :DCB7     (11) OFF SCREEN
434 DAB8 12DD          E?EL    DBL   :DD12     (12) ERROR LINE RUN
435
```

```
436                     * Compile/Run-time errors:
437
438 DABA 47DC         E?OM    DBL   :DC47      (13) OUT OF MEMORY
439 DABC 8ADC         E?TM    DBL   :DC8A      (14) TYPE MISMATCH
440 DABE D6DC         E?LN    DBL   :DCD6      (15) LINE NUMBER OUT OF
441                                              .    RANGE
442 DAC0 38DC         E?SO    DBL   :DC38      (16) STACK OVERFLOW
443
444                     * Compile-time errors:
445
446 DAC2 23DC         E?SN    DBL   :DC23      (17) SYNTAX ERROR
447 DAC4 79DC         E?ID    DBL   :DC79      (18) COMMAND INVALID
448 DAC6 A9DC         E?CN    DBL   :DCA9      (19) CAN'T CONT
449 DAC8 E3DC         E?TC    DBL   :DCE3      (1A) LINE TOO COMPLEX
450 DACA 47DC         E?ST    DBL   :DC47      (1B) OUT OF MEMORY
451                     *
452                     **********************************
453                     * WAIT; part of RTALK (0EC6D) *
454                     **********************************
455                     *
456                     * Entry: HL: Wait time.
457                     *
458 DACC 2B           RTK20   DCX   H          Wait time -1
459 DACD 7C                   MOV   A,H
460 DACE B5                   ORA   L
461 DACF C2CCDA               JNZ   :DACC      If not ready
462 DAD2 EB                   XCHG             Parameter pntr in HL
463 DAD3 C9                   RET
464                     *
465                     *
466                     *
467 DAD4                      END
```

```
****************************
* S Y M B O L   T A B L E *
****************************
```

CKI10	D9C2	CLKEI	D9DB	CLKINT	D9A9	E?BS	DA9E
E?CN	DAC6	E?DO	DAA0	E?EL	DAB8	E?ID	DAC4
E?IN	DAA8	E?LN	DABE	E?LO0	DAAA	E?LO1	DAAC
E?LO2	DAAE	E?LO3	DAB0	E?LS	DAA4	E?NC	DAB4
E?NF	DA94	E?OD	DA98	E?OF	DAB6	E?OM	DABA
E?OS	DAA2	E?OV	DA9A	E?RA	DAA6	E?RG	DA96
E?SN	DAC2	E?SO	DAC0	E?ST	DACA	E?TC	DAC8
E?TM	DABC	E?UA	DAB2	E?US	DA9C	ERITB	DA94
ERRBS	DA29	ERRCO	DA64	ERRDO	DA24	ERREL	DA33
ERRLS	DA38	ERRMS	DA50	ERRNF	DA2E	ERROM	DA10
ERROR	D9F5	ERROV	DA1F	ERRRA	DA15	ERRRU	DA3D
ERRSN	DA0B	ERRTM	DA1A	ERT10	DA4D	INTCH	D977
INTINI	D8FB	INTRM	D9CD	INTSU	D91E	ITMPL	D96B
KBD1	D973	KBEI	D988	KBIS	D99D	MIL10	DA8C
MSGIL	DA75	RTK20	DACC	SNDDI	D98F	SNDEI	D996
SNDIS	D9A3	SPINT	D9E2	VCS10	D94C	VCS20	D963
VECSU	D949						

```
002                             ORG   :DAD4
003                     *
004                     *
005                     *
006                     *  ================
007                     *** PRINT ROUTINES ***
008                     *  ================
009                     *
010                     *
011                     ******************
012                     * PRINT MESSAGE *
013                     ******************
014                     *
015                     * Prints a message, which may include internal
016                     * references to other submessages.
017                     *
018                     * Entry: Pointer to message in HL.
019                     * Exit:  Pointer after string in HL. Other
020                     *        registers preserved.
021                     *
022                     * Message format: A series of bytes:
023                     *        00     End of string.
024                     *        01-7F Printed character.
025                     *        >= 80 bit 14 set:
026                     *              bits 0-13 are offset in program of
027                     *              a message (char. terminated by a 0).
028                     *              bit 14 unset:
029                     *              bits 0-13 are offset in program of
030                     *              a string (1 byte length + char.).
031                     *
032 DAD4 F5             PMSG    PUSH  PSW
033 DAD5 7E             PMS10   MOV   A,M         Get character
034 DAD6 23                     INX   H           Points to next char.
035 DAD7 B7             PMS15   ORA   A           Check char.
036 DAD8 CAE4DA                 JZ    :DAE4       If end message
037 DADB FAE6DA                 JM    :DAE6       If submessage reference
038 DADE CD60DD                 CALL  :DD60       Print character in A
039 DAE1 C3D5DA                 JMP   :DAD5       Next
040                     *
041 DAE4 F1             PMS20   POP   PSW         End message
042 DAE5 C9                     RET
043
044                     * Submessage reference:
045
046 DAE6 FEC0           PMS30   CPI   :C0
047 DAE8 E5                     PUSH  H
048 DAE9 6E                     MOV   L,M         Lobyte in L
049 DAEA DAF6DA                 JC    :DAF6       If reference to string
050 DAED 67                     MOV   H,A         BASE is at location C000
051 DAEE CDD4DA                 CALL  :DAD4       Print submessage
052 DAF1 E1             PMS35   POP   H
053 DAF2 23                     INX   H
054 DAF3 C3D5DA                 JMP   :DAD5       Next character
055
056                     * String reference:
057
058 DAF6 C640           PMS40   ADI   :40         (BASE SHR 8) - #80
059 DAF8 67                     MOV   H,A         Hibyte in H
060 DAF9 CD32DB                 CALL  :DB32       Print substring pntd by HL
061 DAFC C3F1DA                 JMP   :DAF1       Next character
062                     *
063                     *
```

```
                        ********************************************
                        * PRINT MESSAGE POINTED TO BY NEXT 2 BYTES *
                        ********************************************
                        *
                        * Entry: Top of stack points to the address where
                        *        the address of the message can be found.
                        * Exit:  AFBCDEHL preserved.
                        *        Returnaddress on stack.
                        *
DAFF E3         PMSGR   XTHL            Get pntr from stack
DB00 D5                 PUSH  D
DB01 5E                 MOV   E,M       Get lobyte address
DB02 23                 INX   H
DB03 56                 MOV   D,M       Get hibyte address
DB04 23                 INX   H
DB05 EB                 XCHG            Addr message in HL
DB06 CDD4DA             CALL  :DAD4     Print message
DB09 EB                 XCHG            HL pnts after message pntr
DB0A D1                 POP   D
DB0B E3                 XTHL            Restore stack
DB0C C9                 RET
                        *
                        ************************
                        * CURSOR TO NEXT FIELD *
                        ************************
                        *
                        * Part of 'run PRINT' (0E2B3).
                        * Moves the cursor to a new number output field.
                        * The field size is 12 character positions; field
                        * positions: 0,12,24,36,48.
                        *
DB0D EF         PSKP    RST   5         Ask cursor position
DB0E 0C                 DATA  :0C       and size character screen
DB0F 7D                 MOV   A,L       X-coord in L
DB10 FE30               CPI   :30       Already past last field?
DB12 D227DB             JNC   :DB27     Then print car.ret, abort
DB15 D60C       PSK10   SUI   :0C       ) Minus 12 untill underflow
DB17 D215DB             JNC   :DB15     )
DB1A 2F         PSK15   CMA             Restore pos. value
DB1B 3C                 INR   A

                        * Entry from 'SPC' function:

DB1C 57         PSK20   MOV   D,A       Store nr of spaces required
DB1D 3E20       PSK30   MVI   A,:20
DB1F CD60DD             CALL  :DD60     Print space
DB22 15                 DCR   D         Ready?
DB23 C21DDB             JNZ   :DB1D     Next space if not
DB26 C9                 RET
                        *
DB27 C35EDD     PSK40   JMP   :DD5E     Print car.ret
                        *
                        ********************
                        * CURSOR TO TAB(8) *
                        ********************
                        *
                        * Part of 'List current line' (0ECB3).
                        * Moves cursor to column 8 after linenumber.
                        *
                        * Exit: BC preserved. AFDEHL corrupted.
                        *
```

```
126                     SCTAB
127 DB2A EF             PTAB    RST   5          Ask cursor position and
128 DB2B 0C                     DATA  :0C        size character screen
129 DB2C 7D                     MOV   A,L        X-coord after linenr in A
130 DB2D D606                   SUI   :06        Tab must be 8
131 DB2F C31ADB                 JMP   :DB1A      Print additional spaces
132                     *
133                     ****************
134                     * PRINT STRING *
135                     ****************
136                     *
137                     * Prints a string of characters pointed to by HL.
138                     *
139                     * Entry: HL points to string.
140                     * Exit:  HL points after string.
141                     *        Other registers preserved.
142                     *
143                     * String format:
144                     *        1 byte length (0 = no data).
145                     *        N bytes data.
146                     *
147                     SCSTR
148 DB32 F5             PSTR    PUSH  PSW
149 DB33 C5                     PUSH  B
150 DB34 46                     MOV   B,M        String length in B
151 DB35 23             PST10   INX   H
152 DB36 78             PST20   MOV   A,B        Get length
153 DB37 D601                   SUI   :01        Minus 1
154 DB39 47                     MOV   B,A        Nr. char still to be printed
155 DB3A 7E                     MOV   A,M        Get char
156 DB3B D495D6                 CNC   :D695      Print character if not ready
157 DB3E D235DB                 JNC   :DB35      Next one if not ready
158 DB41 C1                     POP   B
159 DB42 F1                     POP   PSW
160 DB43 C9                     RET
161                     *
162                     ************************
163                     * PRINT STRING MESSAGE *
164                     ************************
165                     *
166                     * Prints a string of characters, pointed to by
167                     * HL, length in A.
168                     *
169                     * Entry: HL: Points to string.
170                     *         A:  Number of characters.
171                     * Exit:  HL: Points after string.
172                     *        AFBCDE preserved.
173                     *
174                     SCSTM
175 DB44 F5             PSTRM   PUSH  PSW
176 DB45 C5                     PUSH  B
177 DB46 47                     MOV   B,A        String length in B
178 DB47 C336DB                 JMP   :DB36      Print string
179                     *
180                     **********************
181                     * PRINT A HEX NUMBER *
182                     **********************
183                     *
184                     * Converts MACC to hex in DECBUF and print it.
185                     *
186                     * Exit: HL points after string in DECBUF.
187                     *       BCDE preserved. AF corrupted.
```

```
                        *
DB4A CD2DC0     PHEX    CALL  :C02D     Convert MACC for hex output
DB4D 2A33C0     PGP     LHLD  :C033     Get addr DECBUF
DB50 C332DB             JMP   :DB32     Print string in DECBUF
                        *
                        ****************************
                        * PRINT A INTEGER NUMBER *
                        ****************************
                        *
                        * Prints an integer number from MACC.
                        *
DB53 CD5FDB     PINT    CALL  :DB5F     Convert MACC for output
DB56 C34DDB             JMP   :DB4D     Print contents DECBUF
                        *
                        ***********************************
                        * PRINT A FLOATING POINT NUMBER *
                        ***********************************
                        *
                        * Prints a FPT number from MACC.
                        *
DB59 CD9BCE     PFPT    CALL  :CE9B     Convert MACC for output
DB5C C34DDB             JMP   :DB4D     Print contents DECBUF
                        *
                        ****************************************
                        * CONVERT MACC FOR FIXED POINT OUTPUT *
                        ****************************************
                        *
                        * Places ASCII-equivalent in 00E4-F0, digits before
                        * decimal point in 00F1, length in 00E3.
                        *
                        * Exit: A: Number of digits.
                        *       BCDEHL preserved.
                        *
DB5F CD27C0     IBCP    CALL  :C027     Convert INT for output
DB62 C5                 PUSH  B
DB63 0600               MVI   B,:00     Can trim last dec. place
DB65 CD30C0     BPP     CALL  :C030     Tidy up into external form
DB68 C1                 POP   B
DB69 C9                 RET
                        *
                        *********************************
                        * (Not used, replaced by CE9B) *
                        *********************************
                        *
DB6A 0601       LD216   MVI   B,:01     ) Part of a previous version
DB6C C364DB             JMP   :DB64     )
                        *
                        *
DB6F                    END
```

```
****************************
* S Y M B O L   T A B L E *
****************************

BPP     DB65    IBCP    DB5F    LD216   DB6A    PFPT    DB59
PGP     DB4D    PHEX    DB4A    PINT    DB53    PMS10   DAD5
PMS15   DAD7    PMS20   DAE4    PMS30   DAE6    PMS35   DAF1
PMS40   DAF6    PMSG    DAD4    PMSGR   DAFF    PSK10   DB15
PSK15   DB1A    PSK20   DB1C    PSK30   DB1D    PSK40   DB27
PSKP    DB0D    PST10   DB35    PST20   DB36    PSTR    DB32
PSTRM   DB44    PTAB    DB2A    SCSTM   DB44    SCSTR   DB32
SCTAB   DB2A
```

```
002                             ORG   :DB6F
003                     *
004                     *
005                     *
006                     ***********************************
007                     * STRINGS FOR MACHINE MESSAGES *
008                     ***********************************
009                     *
010                     * The machine messages exist partly from strings,
011                     * partly from subreferences to other strings.
012                     * The subreferences can be:
013                     *     - An address where another string can be found.
014                     *     - An offset with base at C000 to the other
015                     *       string.
016                     * The messages are ended with 00.
017                     * 20 is space, 0D is carriage return.
018                     *
019                     RMS01
020 DB6F 53             MSG01   DATA  :53       S
021 DB70 4F                     DATA  :4F       O
022 DB71 4D                     DATA  :4D       M
023 DB72 45                     DATA  :45       E
024 DB73 20                     DATA  :20
025 DB74 8CA5                   DATA  :8C,:A5   INPUT
026 DB76 20                     DATA  :20
027 DB77 49                     DATA  :49       I
028 DB78 47                     DATA  :47       G
029 DB79 4E                     DATA  :4E       N
030 DB7A 4F                     DATA  :4F       O
031 DB7B 52                     DATA  :52       R
032 DB7C 45                     DATA  :45       E
033 DB7D 44                     DATA  :44       D
034 DB7E 00                     DATA  :00
035                     *
036 DB7F 20             MSG02   DATA  :20
037 DB80 49                     DATA  :49       I
038 DB81 4E                     DATA  :4E       N
039 DB82 20                     DATA  :20
040 DB83 4C             MLINE   DATA  :4C       L
041 DB84 49                     DATA  :49       I
042 DB85 4E                     DATA  :4E       N
043 DB86 45                     DATA  :45       E
044 DB87 20                     DATA  :20
045 DB88 00                     DATA  :00
046                     *
047 DB89 DC0D           MSG03   DATA  :DC,:0D   OUT OF
048 DB8B 8E56                   DATA  :8E,:56   SPACE
049 DB8D 20                     DATA  :20
050 DB8E 8CD2                   DATA  :8C,:D2   FOR
051 DB90 D88D                   DATA  :D8,:8D   MODE
052 DB92 00                     DATA  :00
053                     *
054 DB93 20             MSG04   DATA  :20
055 DB94 52                     DATA  :52       R
056 DB95 45                     DATA  :45       E
057 DB96 DBF7                   DATA  :DB,:F7   TYPE
058 DB98 DB83           MLINR   DATA  :DB,:83   LINE
059 DB9A 0D                     DATA  :0D
060 DB9B 00                     DATA  :00
061                     *
062 DB9C 0D             MSG15   DATA  :0D
063 DB9D 53                     DATA  :53       S
```

```
064 DB9E 45                    DATA  :45        E
065 DB9F 54                    DATA  :54        T
066 DBA0 20                    DATA  :20
067 DBA1 52                    DATA  :52        R
068 DBA2 45                    DATA  :45        E
069 DBA3 43                    DATA  :43        C
070 DBA4 4F                    DATA  :4F        O
071 DBA5 52                    DATA  :52        R
072 DBA6 44                    DATA  :44        D
073 DBA7 2C                    DATA  :2C        ,
074 DBA8 DBB0          MSG05   DATA  :DB,:B0    START TAPE
075 DBAA 2C                    DATA  :2C        ,
076 DBAB DBF7                  DATA  :DB,:F7    TYPE
077 DBAD 8E56                  DATA  :8E,:56    SPACE
078 DBAF 00                    DATA  :00
079                    *
080 DBB0 53            MSG06   DATA  :53        S
081 DBB1 54                    DATA  :54        T
082 DBB2 41                    DATA  :41        A
083 DBB3 52                    DATA  :52        R
084 DBB4 54                    DATA  :54        T
085 DBB5 DBFD                  DATA  :DB,:FD    TAPE
086 DBB7 00                    DATA  :00
087                    *
088 DBB8 0D            MSG07   DATA  :0D
089 DBB9 2A                    DATA  :2A        *
090 DBBA 2A                    DATA  :2A        *
091 DBBB 2A                    DATA  :2A        *
092 DBBC DBE0                  DATA  :DB,:E0    BREAK
093 DBBE 0D                    DATA  :0D
094 DBBF 00                    DATA  :00
095                    *
096 DBC0 20            MSG17   DATA  :20
097 DBC1 4F                    DATA  :4F        O
098 DBC2 4B                    DATA  :4B        K
099 DBC3 0D                    DATA  :0D
100 DBC4 00                    DATA  :00
101                    *
102 DBC5 0D            MSG09   DATA  :0D
103 DBC6 DBE0                  DATA  :DB,:E0    BREAK
104 DBC8 00                    DATA  :00
105                    *
106 DBC9 8BCE          MSG10   DATA  :8B,:CE    STOP
107 DBCB 50                    DATA  :50        P
108 DBCC 45                    DATA  :45        E
109 DBCD 44                    DATA  :44        D
110 DBCE 00                    DATA  :00
111                    *
112 DBCF 8BD6          MSG11   DATA  :8B,:D6    END
113 DBD1 20                    DATA  :20
114 DBD2 50                    DATA  :50        P
115 DBD3 52                    DATA  :52        R
116 DBD4 4F                    DATA  :4F        O
117 DBD5 47                    DATA  :47        G
118 DBD6 52                    DATA  :52        R
119 DBD7 41                    DATA  :41        A
120 DBD8 4D                    DATA  :4D        M
121 DBD9 0D                    DATA  :0D
122 DBDA 00                    DATA  :00
123                    *
124 DBDB 20            MSG14   DATA  :20
    DBDC 42                    DATA  :42        B
```

```
126 DBDD 41                 DATA  :41       A
127 DBDE 44                 DATA  :44       D
128 DBDF 00                 DATA  :00
129                 *
130 DBE0 42         MBREAK  DATA  :42       B
131 DBE1 52                 DATA  :52       R
132 DBE2 45                 DATA  :45       E
133 DBE3 41                 DATA  :41       A
134 DBE4 4B                 DATA  :4B       K
135 DBE5 00                 DATA  :00
136                 *
137 DBE6 20         MWITHO  DATA  :20
138 DBE7 57                 DATA  :57       W
139 DBE8 49                 DATA  :49       I
140 DBE9 54                 DATA  :54       T
141 DBEA 48                 DATA  :48       H
142 DBEB 4F                 DATA  :4F       O
143 DBEC 55                 DATA  :55       U
144 DBED 54                 DATA  :54       T
145 DBEE 20                 DATA  :20
146 DBEF 00                 DATA  :00
147                 *
148 DBF0 53         MSTRIN  DATA  :53       S
149 DBF1 54                 DATA  :54       T
150 DBF2 52                 DATA  :52       R
151 DBF3 49         MING    DATA  :49       I
152 DBF4 4E                 DATA  :4E       N
153 DBF5 47                 DATA  :47       G
154 DBF6 00                 DATA  :00
155                 *
156 DBF7 54         MTYPE   DATA  :54       T
157 DBF8 59                 DATA  :59       Y
158 DBF9 50                 DATA  :50       P
159 DBFA 45                 DATA  :45       E
160 DBFB 20                 DATA  :20
161 DBFC 00                 DATA  :00
162                 *
163 DBFD 20         MTAPE   DATA  :20
164 DBFE 54                 DATA  :54       T
165 DBFF 41                 DATA  :41       A
166 DC00 50                 DATA  :50       P
167 DC01 45                 DATA  :45       E
168 DC02 00                 DATA  :00
169                 *
170 DC03 55         MUNDF   DATA  :55       U
171 DC04 4E                 DATA  :4E       N
172 DC05 44                 DATA  :44       D
173 DC06 45                 DATA  :45       E
174 DC07 46                 DATA  :46       F
175 DC08 49                 DATA  :49       I
176 DC09 4E                 DATA  :4E       N
177 DC0A 45                 DATA  :45       E
178 DC0B 44                 DATA  :44       D
179 DC0C 00                 DATA  :00
180                 *
181 DC0D 4F         MOUTOF  DATA  :4F       O
182 DC0E 55                 DATA  :55       U
183 DC0F 54                 DATA  :54       T
184 DC10 20                 DATA  :20
185 DC11 4F                 DATA  :4F       O
186 DC12 46                 DATA  :46       F
187 DC13 20                 DATA  :20
```

```
188 DC14 00                 DATA  :00
189                     *
190 DC15 20             MERROR  DATA  :20
191 DC16 45                     DATA  :45       E
192 DC17 52                     DATA  :52       R
193 DC18 52                     DATA  :52       R
194 DC19 4F                     DATA  :4F       O
195 DC1A 52                     DATA  :52       R
196 DC1B 00                     DATA  :00
197                     *
198                     ***************************
199                     * STRINGS ERROR MESSAGES *
200                     ***************************
201                     *
202                     * Comments: See strings machine messages (DB6F).
203                     *
204 DC1C 8CD9           ERMNF   DATA  :8C,:D9   NEXT
205 DC1E DBE6                   DATA  :DB,:E6   WITHOUT
206 DC20 8CD2                   DATA  :8C,:D2   FOR
207 DC22 00                     DATA  :00
208                     *
209 DC23 53             ERMSN   DATA  :53       S
210 DC24 59                     DATA  :59       Y
211 DC25 4E                     DATA  :4E       N
212 DC26 54                     DATA  :54       T
213 DC27 41                     DATA  :41       A
214 DC28 58                     DATA  :58       X
215 DC29 DC15                   DATA  :DC,:15   ERROR
216 DC2B 00                     DATA  :00
217                     *
218 DC2C 8BE8           ERMRG   DATA  :8B,:E8   RETURN
219 DC2E DBE6                   DATA  :DB,:E6   WITHOUT
220 DC30 8C01                   DATA  :8C,:01   GOSUB
221 DC32 00                     DATA  :00
222                     *
223 DC33 DC0D           ERMOD   DATA  :DC,:0D   OUT OF
224 DC35 8CAE                   DATA  :8C,:AE   DATA
225 DC37 00                     DATA  :00
226                     *
227 DC38 53             ERMSO   DATA  :53       S
228 DC39 54                     DATA  :54       T
229 DC3A 41                     DATA  :41       A
230 DC3B 43                     DATA  :43       C
231 DC3C 4B                     DATA  :4B       K
232 DC3D 20                     DATA  :20
233 DC3E 4F             ERMOV   DATA  :4F       O
234 DC3F 56                     DATA  :56       V
235 DC40 45                     DATA  :45       E
236 DC41 52                     DATA  :52       R
237 DC42 46                     DATA  :46       F
238 DC43 4C                     DATA  :4C       L
239 DC44 4F                     DATA  :4F       O
240 DC45 57                     DATA  :57       W
241 DC46 00                     DATA  :00
242                     *
243 DC47 DC0D           ERMOM   DATA  :DC,:0D   OUT OF
244 DC49 4D                     DATA  :4D       M
245 DC4A 45                     DATA  :45       E
246 DC4B 4D                     DATA  :4D       M
247 DC4C 4F                     DATA  :4F       O
248 DC4D 52                     DATA  :52       R
249 DC4E 59                     DATA  :59       Y
```

```
250 DC4F 00                 DATA  :00
251                 *
252 DC50 DC03       ERMUS   DATA  :DC,:03   UNDEFINED
253 DC52 20                 DATA  :20
254 DC53 DB53       MLINN   DATA  :DB,:53   LINE
255 DC55 4E         MNUMBE  DATA  :4E       N
256 DC56 55                 DATA  :55       U
257 DC57 4D                 DATA  :4D       M
258 DC58 42                 DATA  :42       B
259 DC59 45                 DATA  :45       E
260 DC5A 52                 DATA  :52       R
261 DC5B 00                 DATA  :00
262                 *
263 DC5C 53         ERMBS   DATA  :53       S
264 DC5D 55                 DATA  :55       U
265 DC5E 42                 DATA  :42       B
266 DC5F 53                 DATA  :53       S
267 DC60 43                 DATA  :43       C
268 DC61 52                 DATA  :52       R
269 DC62 49                 DATA  :49       I
270 DC63 50                 DATA  :50       P
271 DC64 54                 DATA  :54       T
272 DC65 DC15               DATA  :DC,:15   ERROR
273 DC67 00                 DATA  :00
274                 *
275 DC68 44         ERMDO   DATA  :44       D
276 DC69 49                 DATA  :49       I
277 DC6A 56                 DATA  :56       V
278 DC6B 49                 DATA  :49       I
279 DC6C 53                 DATA  :53       S
280 DC6D 49                 DATA  :49       I
281 DC6E 4F                 DATA  :4F       O
282 DC6F 4E                 DATA  :4E       N
283 DC70 20                 DATA  :20
284 DC71 42                 DATA  :42       B
285 DC72 59                 DATA  :59       Y
286 DC73 20                 DATA  :20
287 DC74 5A                 DATA  :5A       Z
288 DC75 45                 DATA  :45       E
289 DC76 52                 DATA  :52       R
290 DC77 4F                 DATA  :4F       O
291 DC78 00                 DATA  :00
292                 *
293 DC79 43         ERMID   DATA  :43       C
294 DC7A 4F                 DATA  :4F       O
295 DC7B 4D                 DATA  :4D       M
296 DC7C 4D                 DATA  :4D       M
297 DC7D 41                 DATA  :41       A
298 DC7E 4E                 DATA  :4E       N
299 DC7F 44                 DATA  :44       D
300 DC80 20                 DATA  :20
301 DC81 49         MINVAL  DATA  :49       I
302 DC82 4E                 DATA  :4E       N
303 DC83 56                 DATA  :56       V
304 DC84 41                 DATA  :41       A
305 DC85 4C                 DATA  :4C       L
306 DC86 49                 DATA  :49       I
307 DC87 44                 DATA  :44       D
308 DC88 20                 DATA  :20
309 DC89 00                 DATA  :00
310                 *
311 DC8A DBF7       ERMTM   DATA  :DB,:F7   TYPE
```

```
312 DC8C 4D                     DATA  :4D       M
313 DC8D 49                     DATA  :49       I
314 DC8E 53                     DATA  :53       S
315 DC8F 4D                     DATA  :4D       M
316 DC90 41                     DATA  :41       A
317 DC91 54                     DATA  :54       T
318 DC92 43                     DATA  :43       C
319 DC93 48                     DATA  :48       H
320 DC94 00                     DATA  :00
321                     *
322 DC95 DC0D           ERMOS   DATA  :DC,:0D   OUT OF
323 DC97 DBF0                   DATA  :DB,:F0   STRING
324 DC99 20                     DATA  :20
325 DC9A 8E56                   DATA  :8E,:56   SPACE
326 DC9C 00                     DATA  :00
327                     *
328 DC9D DBF0           ERMLS   DATA  :DB,:F0   STRING
329 DC9F 20                     DATA  :20
330 DCA0 54                     DATA  :54       T
331 DCA1 4F                     DATA  :4F       O
332 DCA2 4F                     DATA  :4F       O
333 DCA3 20                     DATA  :20
334 DCA4 4C                     DATA  :4C       L
335 DCA5 4F                     DATA  :4F       O
336 DCA6 4E                     DATA  :4E       N
337 DCA7 47                     DATA  :47       G
338 DCA8 00                     DATA  :00
339                     *
340 DCA9 43             ERMCN   DATA  :43       C
341 DCAA 41                     DATA  :41       A
342 DCAB 4E                     DATA  :4E       N
343 DCAC 27                     DATA  :27       '
344 DCAD 54                     DATA  :54       T
345 DCAE 20                     DATA  :20
346 DCAF 8BC6                   DATA  :8B,:C6   CONT
347 DCB1 00                     DATA  :00
348                     *
349 DCB2 DC81           ERMIN   DATA  :DC,:81   INVALID
350 DCB4 DC55                   DATA  :DC,:55   NUMBER
351 DCB6 00                     DATA  :00
352                     *
353 DCB7 4F             ERMOF   DATA  :4F       O
354 DCB8 46                     DATA  :46       F
355 DCB9 46                     DATA  :46       F
356 DCBA 20                     DATA  :20
357 DCBB 53                     DATA  :53       S
358 DCBC 43                     DATA  :43       C
359 DCBD 52                     DATA  :52       R
360 DCBE 45                     DATA  :45       E
361 DCBF 45                     DATA  :45       E
362 DCC0 4E                     DATA  :4E       N
363 DCC1 00                     DATA  :00
364                     *
365 DCC2 43             ERMNC   DATA  :43       C
366 DCC3 4F                     DATA  :4F       O
367 DCC4 4C                     DATA  :4C       L
368 DCC5 4F                     DATA  :4F       O
369 DCC6 52                     DATA  :52       R
370 DCC7 20                     DATA  :20
371 DCC8 4E                     DATA  :4E       N
372 DCC9 4F                     DATA  :4F       O
373 DCCA 54                     DATA  :54       T
```

```
374 DCCB 20                     DATA  :20
375 DCCC 41                     DATA  :41       A
376 DCCD 56                     DATA  :56       V
377 DCCE 41                     DATA  :41       A
378 DCCF 49                     DATA  :49       I
379 DCD0 4C                     DATA  :4C       L
380 DCD1 41                     DATA  :41       A
381 DCD2 42                     DATA  :42       B
382 DCD3 4C                     DATA  :4C       L
383 DCD4 45                     DATA  :45       E
384 DCD5 00                     DATA  :00
385                     *
386 DCD6 DB83           ERMLN   DATA  :DB,:83   LINE
387 DCD8 DC55           ERMNA   DATA  :DC,:55   NUMBER
388 DCDA 20                     DATA  :20
389 DCDB DC0D           MSGOR   DATA  :DC,:0D   OUT OF
390 DCDD 52                     DATA  :52       R
391 DCDE 41                     DATA  :41       A
392 DCDF 4E                     DATA  :4E       N
393 DCE0 47                     DATA  :47       G
394 DCE1 45                     DATA  :45       E
395 DCE2 00                     DATA  :00
396                     *
397 DCE3 DB83           ERMTC   DATA  :DB,:83   LINE
398 DCE5 54                     DATA  :54       T
399 DCE6 4F                     DATA  :4F       O
400 DCE7 4F                     DATA  :4F       O
401 DCE8 20                     DATA  :20
402 DCE9 43                     DATA  :43       C
403 DCEA 4F                     DATA  :4F       O
404 DCEB 4D                     DATA  :4D       M
405 DCEC 50                     DATA  :50       P
406 DCED 4C                     DATA  :4C       L
407 DCEE 45                     DATA  :45       E
408 DCEF 58                     DATA  :58       X
409 DCF0 00                     DATA  :00
410                     *
411 DCF1 DC03           ERMUA   DATA  :DC,:03   UNDEFINED
412 DCF3 20                     DATA  :20
413 DCF4 41                     DATA  :41       A
414 DCF5 52                     DATA  :52       R
415 DCF6 52                     DATA  :52       R
416 DCF7 41                     DATA  :41       A
417 DCF8 59                     DATA  :59       Y
418 DCF9 00                     DATA  :00
419                     *
420 DCFA DD0A           ERML0   DATA  :DD,:0A   LOADING ERROR
421 DCFC 30                     DATA  :30       0
422 DCFD 00                     DATA  :00
423                     *
424 DCFE DD0A           ERML1   DATA  :DD,:0A   LOADING ERROR
425 DD00 31                     DATA  :31       1
426 DD01 00                     DATA  :00
427                     *
428 DD02 DD0A           ERML2   DATA  :DD,:0A   LOADING ERROR
429 DD04 32                     DATA  :32       2
430 DD05 00                     DATA  :00
431                     *
432 DD06 DD0A           ERML3   DATA  :DD,:0A   LOADING ERROR
433 DD08 33                     DATA  :33       3
434 DD09 00                     DATA  :00
435                     *
```

```
436                     ERMLO
437 DD0A 8D1B           MSGL    DATA  :8D,:1B   LOAD
438 DD0C DBF3                   DATA  :DB,:F3   ING
439 DD0E DC15                   DATA  :DC,:15   ERROR
440 DD10 20                     DATA  :20
441 DD11 00                     DATA  :00
442                     *
443 DD12 DC15           ERMEL   DATA  :DC,:15   ERROR
444 DD14 20                     DATA  :20
445 DD15 DB83                   DATA  :DB,:83   LINE
446 DD17 8BF2                   DATA  :8B,:F2   RUN
447 DD19 00                     DATA  :00
448                     *
449                     *
450                     *
451 DD1A                        END
```

```
*****************************
* S Y M B O L   T A B L E *
*****************************
```

ERMBS	DC5C	ERMCN	DCA9	ERMDO	DC68	ERMEL	DD12
ERMID	DC79	ERMIN	DCB2	ERMLO	DCFA	ERML1	DCFE
ERML2	DD02	ERML3	DD06	ERMLN	DCD6	ERMLO	DD0A
ERMLS	DC9D	ERMNA	DCD8	ERMNC	DCC2	ERMNF	DC1C
ERMOD	DC33	ERMOF	DCB7	ERMOM	DC47	ERMOS	DC95
ERMOV	DC3E	ERMRG	DC2C	ERMSN	DC23	ERMSO	DC38
ERMTC	DCE3	ERMTM	DC8A	ERMUA	DCF1	ERMUS	DC50
MBREAK	DBE0	MERROR	DC15	MING	DBF3	MINVAL	DC81
MLINE	DB83	MLINN	DC53	MLINR	DB98	MNUMBE	DC55
MOUTOF	DC0D	MSG01	DB6F	MSG02	DB7F	MSG03	DB89
MSG04	DB93	MSG05	DBA8	MSG06	DBB0	MSG07	DBB8
MSG09	DBC5	MSG10	DBC9	MSG11	DBCF	MSG14	DBDB
MSG15	DB9C	MSG17	DBC0	MSGL	DD0A	MSGOR	DCDB
MSTRIN	DBF0	MTAPE	DBFD	MTYPE	DBF7	MUNDF	DC03
MWITHO	DBE6	RMS01	DB6F				

```
002                             ORG   :DD1A
003                     *
004                     *
005                     *
006                     ********************
007                     * INPUT TEXT LINE *
008                     ********************
009                     *
010                     * Part of 'restart interpreter' (C853).
011                     * Scans keyboard and reads in a line on the current
012                     * screen line, up until car.ret. First prints
013                     * car.ret and a prompt ('*').
014                     * The routine can be aborted on car.ret or Break
015                     * only.
016                     *
017                     * Entry: A: Contains prompt.
018                     * Exit:  CY=1: Break pressed.
019                     *        CY=0: HL:  Address 1st character on line.
020                     *              C=1: Offset 1st significant character
021                     *              ABDEHL preserved.
022                     *
023 DD1A F5             INPL0   PUSH  PSW
024 DD1B CD55DD                CALL  :DD55     Cursor to column 0
025 DD1E F1                    POP   PSW       Get prompt
026 DD1F 37             INPLN   STC             CY=1
027 DD20 F5                    PUSH  PSW
028 DD21 E5                    PUSH  H         Save cursor coord 1st char
029 DD22 CD6ADD         IPL10   CALL  :DD6A     Print prompt
030 DD25 21B902                LXI   H,:02B9
031 DD28 3600                  MVI   M,:00     Enable complete keyb.scan
032 DD2A CDBED6         IPL20   CALL  :D6BE     Get keyb. input
033 DD2D DA49DD                JC    :DD49     Ignore line if Break presse
034 DD30 CA2ADD                JZ    :DD2A     Wait for input
035 DD33 FE20                  CPI   :20       Printable character?
036 DD35 D222DD                JNC   :DD22     Print it and get next one
037 DD38 FE08                  CPI   :08       Backspace
038 DD3A CA22DD                JZ    :DD22     Print it; get next char
039 DD3D FE0D                  CPI   :0D       Car.ret?
040 DD3F C22ADD                JNZ   :DD2A     Get next char if not
041
042                     * Exit on car.ret:
043
044 DD42 35                    DCR   M         Set KBRFL for BREAK only
045 DD43 0E01                  MVI   C,:01
046 DD45 E1             EXIT1   POP   H         Get cursor coord 1st char
047 DD46 F1                    POP   PSW       Get prompt
048 DD47 3F                    CMC             CY=0
049 DD48 C9                    RET
050
051                     * Exit on Break:
052
053 DD49 35             IPL30   DCR   M         Set KBRFL for BREAK only
054 DD4A 3E21                  MVI   A,:21
055 DD4C CD60DD                CALL  :DD60     Print '!'
056 DD4F CD5EDD                CALL  :DD5E     Print car.ret
057 DD52 E1                    POP   H
058 DD53 F1                    POP   PSW       Get prompt, CY=1
059 DD54 C9                    RET
060                     *
061                     **********************
062                     * CURSOR TO COLUMN 0 *
063                     **********************
```

```
                        *
                        * The X-coordinate of the cursor is checked.
                        * If not 0, a car.ret is printed.
                        *
                        * Entry: None.
                        * Exit:  AF corrupted. BCDEHL preserved.
                        *
DD55 D5         COL0    PUSH  D
DD56 E5                 PUSH  H
DD57 EF                 RST   5           Get cursor pos (HL) and size
DD58 0C                 DATA  :0C         char.screen (DE)
DD59 7D                 MOV   A,L         X-coord cursor in A
DD5A E1                 POP   H
DD5B D1                 POP   D
DD5C B7                 ORA   A
DD5D C8                 RZ                Abort if cursor in Col.0
DD5E 3E0D       CRLF    MVI   A,:0D       Else: print CR
                        *
                        * GENERAL OUTPUT ROUTINE:
                        *
                        * Outputs a character in a direction depending
                        * on OTSW (#0131).
                        *
                        * Entry: A: Character to be transmitted.
                        * Exit:  AF corrupted.
                        *
                        SCCHR
DD60 F5         OUTC    PUSH  PSW         Preserve char
DD61 3A3101             LDA   :0131
DD64 FE02               CPI   :02         Check output direction
DD66 D270DD             JNC   :DD70       If to edit buf/DOUTC

                        * To screen/RS232 - OTSW=0/1:

DD69 F1                 POP   PSW         Get char
DD6A EF         COUTC   RST   5           Character to screen
DD6B 03                 DATA  :03
DD6C D442D6             CNC   :D642       Output to RS232 if reqd
DD6F C9                 RET

                        * To DOUTC - OTSW=3:

DD70 00         OTC10   NOP
DD71 C24CD7             JNZ   :D74C       Character to DOUTC

                        * To editbuffer - OTSW=2:

DD74 F1                 POP   PSW         Get char
DD75 F5         OTBIN   PUSH  PSW
DD76 E5                 PUSH  H
DD77 D5                 PUSH  D
DD78 2AA400             LHLD  :00A4       Get edit input pointer
DD7B 77                 MOV   M,A         Byte in edit buffer
DD7C 23                 INX   H
DD7D 22A400             SHLD  :00A4       Update edit input pointer
DD80 EB                 XCHG              in DE
DD81 2AA600             LHLD  :00A6       Get end edit buffer
DD84 CD14DE             CALL  :DE14       Calculate free buffer space
DD87 DA8EDD             JC    :DD8E       If edit buffer full
DD8A D1                 POP   D
DD8B E1                 POP   H
                        OTC99   POP   PSW
```

```
DD8D C9                 RET
                        * If edit buffer full:
                        *
DD8E CDCADE     OTC20   CALL  :DECA     Heap back to right size
DD91 C310DA             JMP   :DA10     Run error 'OUT OF MEMORY'
                        *
                        ********************
                        * OUTPUT TO RS232 *
                        ********************
                        *
                        * Transmits a character to the RS232 interface via
                        * the TICC if the interface is ready for it.
                        * In case of a car.ret, also a line feed is send.
                        *
                        * Entry: A: Character to be transmitted.
                        * Exit:  ABCDEHL preserved, F corrupted.
                        *
DD94 F5         OUTSE   PUSH  PSW       Preserve char
DD95 3A00FD     OTS10   LDA   :FD00
DD98 E608               ANI   :08       Check peripheral ready
DD9A CA95DD             JZ    :DD95     Wait untill ready
DD9D 3AF3FF     OTS20   LDA   :FFF3
DDA0 E610               ANI   :10       Check TICC buffer empty
DDA2 CA9DDD             JZ    :DD9D     Wait untill empty
DDA5 F1                 POP   PSW       Get char
DDA6 32F6FF             STA   :FFF6     Load serial output buffer
DDA9 FE0D               CPI   :0D       Carriage return?
DDAB C0                 RNZ             Ready if not
                        * If car.ret:
                        *
DDAC F5                 PUSH  PSW
DDAD 3E0A               MVI   A,:0A
DDAF CD94DD             CALL  :DD94     Send line feed too
DDB2 F1                 POP   PSW
DDB3 C9                 RET
                        *
                        ********************
                        * INPUT FROM RS232 *
                        ********************
                        *
                        * Gets inputs from RS232 via TICC. Only 7-bit
                        * Ascii-code is accepted.
                        *
                        * Entry: No conditions.
                        * Exit:  A: Character received (0 if nothing).
                        *        BCDEHL preserved.
                        *
                        CINC
DDB4 3AF3FF     INSER   LDA   :FFF3
DDB7 E608               ANI   :08       Check if something received
DDB9 C8                 RZ              Abort if no reception
DDBA 3AF0FF             LDA   :FFF0     Received char in A
DDBD E67F               ANI   :7F       Mask bit 7
DDBF C9                 RET
                        *
                        ***********************************
                        * RS232 FRAME ERROR - (not used) *
                        ***********************************
                        *
                        * Break test for serial input line.
```

```
188                     *
189 DDC0 3AF3FF         BRSER   LDA    :FFF3
190 DDC3 1F                     RAR
191 DDC4 D0                     RNC                 Abort if no break
192 DDC5 3AF3FF         BRS10   LDA    :FFF3        If break: check again
193 DDC8 1F                     RAR
194 DDC9 DAC5DD                 JC     :DDC5        Wait until end of break
195 DDCC 3AF0FF                 LDA    :FFF0        Load received character
196 DDCF 3F                     CMC                 Set CY=1
197 DDD0 C9                     RET
198                     *
199                     *
200                     *  ==========================
201                     *** ENCODING SERVICE ROUTINES ***
202                     *  ==========================
203                     *
204                     *
205                     * The following routines are used both in 'main'
206                     * and 'decode' modules.
207                     *
208                     ************************************************
209                     * GET CHARACTER FROM LINE, NEGLECT TAB + SPACE *
210                     ************************************************
211                     *
212                     * Entry: C: Position on current line.
213                     * Exit:  Character in A; tab and space neglected.
214                     *        C: Points to next character.
215                     *        BDEHL preserved.
216                     *
217 DDD1 0C               IGNBR   INR    C            Pnts to next char
218 DDD2 CDE0DD           IGNB    CALL   :DDE0        Get char from line
219 DDD5 FE20                   CPI    :20          Space?
220 DDD7 CAD1DD                 JZ     :DDD1        Then get next char
221 DDDA FE09                   CPI    :09          Tab?
222 DDDC CAD1DD                 JZ     :DDD1        Then get next char
223 DDDF C9                     RET
224                     *
225                     ****************************
226                     * GET CHARACTER TO ENCODE *
227                     ****************************
228                     *
229                     * Returns a character from some position on the
230                     * current line. The source is determined by EFSW.
231                     *
232                     * Entry: C: Position on current line (max. 219).
233                     * Exit:  EFSW=0  - Keyboard: Char on line pos in A.
234                     *        EFSW>=2 - Edit buf: Char on EFEPT + line
235                     *                            pos in A.
236                     *        EFSW=1  - String:   Idem. If COUNT=line pos
237                     *                            then char is car.ret.
238                     *        F corrupted, BCDEHL preserved.
239                     *
240 DDE0 3A3501           EFETCH  LDA    :0135
241 DDE3 FE01                   CPI    :01          Check input direction
242 DDE5 DAFFDD                 JC     :DDFF        If from keyb/RS232
243 DDE8 C2F4DD                 JNZ    :DDF4        If from edit buffer
244
245                     * If from string:
246
247 DDEB 3A3401                 LDA    :0134        If string: Get COUNT
248 DDEE B9                     CMP    C            COUNT=pos.on curr.line?
249 DDEF 3E0D                   MVI    A,:0D        Then char is car.ret
```

```
188                     *
189 DDC0 3AF3FF         BRSER   LDA    :FFF3
190 DDC3 1F                     RAR
191 DDC4 D0                     RNC                 Abort if no break
192 DDC5 3AF3FF         BRS10   LDA    :FFF3        If break: check again
193 DDC8 1F                     RAR
194 DDC9 DAC5DD                 JC     :DDC5        Wait until end of break
195 DDCC 3AF0FF                 LDA    :FFF0        Load received character
196 DDCF 3F                     CMC                 Set CY=1
197 DDD0 C9                     RET
198                     *
199                     *
200                     *  ==========================
201                     *** ENCODING SERVICE ROUTINES ***
202                     *  ==========================
203                     *
204                     *
205                     * The following routines are used both in 'main'
206                     * and 'decode' modules.
207                     *
208                     ************************************************
209                     * GET CHARACTER FROM LINE, NEGLECT TAB + SPACE *
210                     ************************************************
211                     *
212                     * Entry: C: Position on current line.
213                     * Exit:  Character in A; tab and space neglected.
214                     *        C: Points to next character.
215                     *        BDEHL preserved.
216                     *
217 DDD1 0C               IGNBR   INR    C            Pnts to next char
218 DDD2 CDE0DD           IGNB    CALL   :DDE0        Get char from line
219 DDD5 FE20                   CPI    :20          Space?
220 DDD7 CAD1DD                 JZ     :DDD1        Then get next char
221 DDDA FE09                   CPI    :09          Tab?
222 DDDC CAD1DD                 JZ     :DDD1        Then get next char
223 DDDF C9                     RET
224                     *
225                     ****************************
226                     * GET CHARACTER TO ENCODE *
227                     ****************************
228                     *
229                     * Returns a character from some position on the
230                     * current line. The source is determined by EFSW.
231                     *
232                     * Entry: C: Position on current line (max. 219).
233                     * Exit:  EFSW=0  - Keyboard: Char on line pos in A.
234                     *        EFSW>=2 - Edit buf: Char on EFEPT + line
235                     *                            pos in A.
236                     *        EFSW=1  - String:   Idem. If COUNT=line pos
237                     *                            then char is car.ret.
238                     *        F corrupted, BCDEHL preserved.
239                     *
240 DDE0 3A3501           EFETCH  LDA    :0135
241 DDE3 FE01                   CPI    :01          Check input direction
242 DDE5 DAFFDD                 JC     :DDFF        If from keyb/RS232
243 DDE8 C2F4DD                 JNZ    :DDF4        If from edit buffer
244
245                     * If from string:
246
247 DDEB 3A3401                 LDA    :0134        If string: Get COUNT
248 DDEE B9                     CMP    C            COUNT=pos.on curr.line?
249 DDEF 3E0D                   MVI    A,:0D        Then char is car.ret
```

```
DDF1 CAFEDD             JZ    :DDFE     And abort

                      * Entry if from edit buffer:

DDF4 E5         EFC10   PUSH  H
DDF5 2A3201             LHLD  :0132     Get EFEPT
DDF8 79                 MOV   A,C
DDF9 CD30DE             CALL  :DE30     Add curr.line pos to EFEPT
DDFC 7E                 MOV   A,M       Get character
DDFD E1                 POP   H
DDFE C9         EFC20   RET

                      * If from screen:

DDFF EF         EFC30   RST   5         Get character from line
DE00 15                 DATA  :15
DE01 C9                 RET
                      *
                      *
                      *  ==================================
                      *** SINGLE AND DOUBLE BYTE UTILITIES ***
                      *  ==================================
                      *
                      *
                      ************************************
                      * CHECK IF UPPER CASE CHARACTER *
                      ************************************
                      *
                      * Entry: A: Character to be checked.
                      * Exit:  CY=0: Not upper case.
                      *        CY=1: Upper case.
                      *        ABCDEHL preserved, F corrupted.
                      *
DE02 FE41       ALPHA   CPI   :41       Lowest upper case char
DE04 3F                 CMC
DE05 D0                 RNC
DE06 FE5B               CPI   :5B       First lower case char
DE08 C9                 RET
                      *
                      ************************************************
                      * CHECK IF CHARACTER IS NUMBER OR UPPER CASE *
                      ************************************************
                      *
                      * Entry: A: Character to be checked.
                      * Exit:  CY=0: Not number, not upper case.
                      *        CY=1: Number or upper case.
                      *        ABCDEHL preserved. F corrupted.
                      *
DE09 CD02DE     ALNUM   CALL  :DE02     Check if upper case
DE0C D8                 RC
DE0D FE30       NUMER   CPI   :30       Lowest number
DE0F 3F                 CMC
DE10 D0                 RNC
DE11 FE3A               CPI   :3A       No number anymore
DE13 C9                 RET
                      *
                      **********************
                      * COMPARE HL AND DE *
                      **********************
                      *
                      * Compares HL with DE (HL-DE).
                      *
```

```
DDF1 CAFEDD             JZ    :DDFE     And abort

                      * Entry if from edit buffer:

DDF4 E5         EFC10   PUSH  H
DDF5 2A3201             LHLD  :0132     Get EFEPT
DDF8 79                 MOV   A,C
DDF9 CD30DE             CALL  :DE30     Add curr.line pos to EFEPT
DDFC 7E                 MOV   A,M       Get character
DDFD E1                 POP   H
DDFE C9         EFC20   RET

                      * If from screen:

DDFF EF         EFC30   RST   5         Get character from line
DE00 15                 DATA  :15
DE01 C9                 RET
                      *
                      *
                      *  ==================================
                      *** SINGLE AND DOUBLE BYTE UTILITIES ***
                      *  ==================================
                      *
                      *
                      ************************************
                      * CHECK IF UPPER CASE CHARACTER *
                      ************************************
                      *
                      * Entry: A: Character to be checked.
                      * Exit:  CY=0: Not upper case.
                      *        CY=1: Upper case.
                      *        ABCDEHL preserved, F corrupted.
                      *
DE02 FE41       ALPHA   CPI   :41       Lowest upper case char
DE04 3F                 CMC
DE05 D0                 RNC
DE06 FE5B               CPI   :5B       First lower case char
DE08 C9                 RET
                      *
                      ************************************************
                      * CHECK IF CHARACTER IS NUMBER OR UPPER CASE *
                      ************************************************
                      *
                      * Entry: A: Character to be checked.
                      * Exit:  CY=0: Not number, not upper case.
                      *        CY=1: Number or upper case.
                      *        ABCDEHL preserved. F corrupted.
                      *
DE09 CD02DE     ALNUM   CALL  :DE02     Check if upper case
DE0C D8                 RC
DE0D FE30       NUMER   CPI   :30       Lowest number
DE0F 3F                 CMC
DE10 D0                 RNC
DE11 FE3A               CPI   :3A       No number anymore
DE13 C9                 RET
                      *
                      **********************
                      * COMPARE HL AND DE *
                      **********************
                      *
                      * Compares HL with DE (HL-DE).
                      *
```

```
                        * Exit: DE=HL: Z=1, CY=0.
                        *       DE<HL: Z=0, CY=0.
                        *       DE>HL: Z=0, CY=1.
                        *       BCDEHL preserved, AF corrupted.
                        *
DE14 7C                 COMP    MOV   A,H
DE15 BA                         CMP   D
DE16 C0                         RNZ
DE17 7D                         MOV   A,L
DE18 BB                         CMP   E
DE19 C9                         RET
                        *
                        ********************************
                        * CALCULATE LENGTH OF BLOCK *
                        ********************************
                        *
                        * Sets HL=HL-DE.
                        *
                        * Entry: Startaddress in DE, 1st address after
                        *        block in HL.
                        * Exit:  Length in HL, startaddress in DE.
                        *        If DE>HL, length in 2-complement.
                        *        ABCDE preserved, F as in COMP.
                        *
DE1A C5                 SUBDE   PUSH  B
DE1B F5                         PUSH  PSW
DE1C 7D                         MOV   A,L
DE1D 93                         SUB   E         Calc. difference lowest byte
DE1E 6F                         MOV   L,A
DE1F 7C                         MOV   A,H
DE20 9A                         SBB   D         Calc. diff. highest byte
DE21 67                         MOV   H,A
DE22 C1                         POP   B
DE23 78                         MOV   A,B
DE24 C1                         POP   B
DE25 C9                         RET
                        *
                        ********************************
                        * DOUBLE BYTE TWO COMPLEMENT *
                        ********************************
                        *
                        * Sets HL=-HL.
                        *
                        * Entry: Double byte to be converted in HL.
                        * Exit:  Two complement in HL. ABCDEF preserved.
                        *
DE26 F5                 CMPHL   PUSH  PSW
DE27 7C                         MOV   A,H
DE28 2F                         CMA             Complement H
DE29 67                         MOV   H,A
DE2A 7D                         MOV   A,L
DE2B 2F                         CMA             Complement L
DE2C 6F                         MOV   L,A
DE2D 23                         INX   H         Add 1
DE2E F1                         POP   PSW
DE2F C9                         RET
                        *
                        ***************************
                        * ADD OFF-SET TO ADDRESS *
                        ***************************
                        *
                        * Adds a given offset to a base address (HL=HL+A).
```

```
                        *
                        * Entry: Base in HL, offset in A.
                        * Exit:  HL=HL+A. ABCDE preserved, F corrupted.
                        *
DE30 F5         DADA    PUSH  PSW
DE31 85                 ADD   L
DE32 6F                 MOV   L,A       L=L+A
DE33 7C                 MOV   A,H
DE34 CE00               ACI   :00       Add carry if overflow
DE36 67                 MOV   H,A
DE37 F1                 POP   PSW
DE38 C9                 RET
                        *
                        ************************************
                        * CALCULATE ADDRESS AFTER STRING *
                        ************************************
                        *
                        * Sets HL=HL+M+1.
                        *
                        * Entry: HL points to 1st byte of string (length
                        *        byte).
                        * Exit:  HL points to first byte after string.
                        *        AFBCDE preserved.
                        *
DE39 F5         DADM    PUSH  PSW
DE3A 7E                 MOV   A,M       Get length of string
DE3B 23                 INX   H         HL: addr. 1st char. byte
DE3C CD30DE             CALL  :DE30     Calc addr after string
DE3F F1                 POP   PSW
DE40 C9                 RET
                        *
                        ******************
                        * DELAY ROUTINE *
                        ******************
                        *
                        * Runs a fixed delay loop of 665 msec. If
                        * interrupts are enabled, the delay will be
                        * approx. 750 msec.
                        * HL is loaded with FFFF, and then decremented.
                        *
                        * Exit: ABCDEHL preserved; F corrupted.
                        *
DE41 E5         DELAY   PUSH  H
DE42 D5                 PUSH  D
DE43 21FFFF             LXI   H,:FFFF   Init. delay value
DE46 54                 MOV   D,H
DE47 5D                 MOV   E,L
DE48 19         DLY10   DAD   D
DE49 DA48DE             JC    :DE48     Repeat if not ready
DE4C D1                 POP   D
DE4D E1                 POP   H
DE4E C9                 RET
                        *
                        ************************
                        * DATA BLOCK TRANSFER *
                        ************************
                        *
                        * Moves a block of data starting at (DE) and
                        * ending at (HL)-1 to (BC).
                        *
                        * Entry: DE: Startaddr. source bank.
                        *        BC: Startaddr. destination bank.
```

```
                        *         HL: Points after end source bank.
                        * Exit:   AF preserved, BCDEHL corrupted.
                        *
DE4F F5         MOVE    PUSH  PSW
DE50 E5                 PUSH  H
DE51 CD1ADE             CALL  :DE1A     Calc. length source bank
DE54 79                 MOV   A,C
DE55 93                 SUB   E
DE56 78                 MOV   A,B
DE57 9A                 SBB   D
DE58 DA6CDE             JC    :DE6C     If destination addr. is
                                        lower than source addr.

                        * Destination address > source address:

DE5B 54                 MOV   D,H
DE5C 5D                 MOV   E,L       Save length in DE
DE5D 09                 DAD   B         Highest dest.addr. in HL
DE5E C1                 POP   B
DE5F 7A         MOV10   MOV   A,D       Check if ready
DE60 B3                 ORA   E
DE61 CA7ADE             JZ    :DE7A     Then abort
DE64 1B                 DCX   D
DE65 2B                 DCX   H
DE66 0B                 DCX   B
DE67 0A                 LDAX  B         Get byte to be transferred
DE68 77                 MOV   M,A       Transfer it
DE69 C35FDE             JMP   :DE5F     Next one

                        * Destination address < source address:

DE6C 7C         MOV20   MOV   A,H
DE6D B5                 ORA   L
DE6E CA79DE             JZ    :DE79     Abort if ready
DE71 2B                 DCX   H
DE72 1A                 LDAX  D         Get byte to be transferred
DE73 02                 STAX  B         Transfer it
DE74 13                 INX   D
DE75 03                 INX   B
DE76 C36CDE             JMP   :DE6C     Next byte

                        * If ready:

DE79 E1         MOV30   POP   H
DE7A F1         MOV40   POP   PSW
DE7B C9                 RET
                        *
                        ************************************
                        * FILL BANK WITH IDENTICAL DATA *
                        ************************************
                        *
                        * Fills an area of memory with a constant.
                        *
                        * Entry: DE: Startaddr. of bank.
                        *        HL: Points after bank.
                        *        A:  Data to be loaded into bank.
                        * Exit:  DE: Points after bank.
                        *        BCHL preserved, AF corrupted.
                        *
DE7C C5         FILL    PUSH  B
DE7D 47                 MOV   B,A       Save data in B
DE7E CD14DE     FIL10   CALL  :DE14     Check if bank full
```

```
498 DE81 CA8DDE            JZ    :DE8D     Abort if ready
499 DE84 DA8DDE            JC    :DE8D     Abort if DE>HL
500 DE87 78                MOV   A,B       Get data
501 DE88 12                STAX  D         and store it
502 DE89 13                INX   D
503 DE8A C37EDE            JMP   :DE7E     Next addr.
504 DE8D C1        FIL20   POP   B
505 DE8E C9                RET
506                *
507                **********************
508                * MULTIPLY HL BY A *
509                **********************
510                *
511                * Multiplies a 16-bit value by a 8-bit value.
512                *
513                * Entry: HL: 16-bit value.
514                *         A:  8-bit value.
515                * Exit:  CY=0: Result in HL.
516                *        CY=1: Overflow.
517                *        ABCDE preserved.
518                *
519 DE8F 37        HLMUL   STC
520 DE90 F5                PUSH  PSW
521 DE91 D5                PUSH  D
522 DE92 EB                XCHG
523 DE93 210000            LXI   H,:0000   Init. result
524 DE96 B7        HLM10   ORA   A
525 DE97 1F                RAR             Next bit of multiplier
526 DE98 D29FDE            JNC   :DE9F     Jump if bit is 0
527 DE9B 19                DAD   D         Add 1* HL if bit is 1
528 DE9C DAADDE            JC    :DEAD     Abort if overflow
529 DE9F B7        HLM20   ORA   A
530 DEA0 CAB1DE            JZ    :DEB1     Abort if ready
531 DEA3 EB                XCHG
532 DEA4 29                DAD   H         Multiply *2
533 DEA5 EB                XCHG
534 DEA6 D296DE            JNC   :DE96     Again if no overflow
535 DEA9 00                NOP
536 DEAA 00                NOP
537 DEAB 00                NOP
538 DEAC 00                NOP
539 DEAD 00        HLM90   NOP
540 DEAE D1                POP   D
541 DEAF F1                POP   PSW       Error exit if overflow
542 DEB0 C9                RET
543 DEB1 D1        HLM99   POP   D
544 DEB2 F1                POP   PSW
545 DEB3 3F                CMC             No error exit
546 DEB4 C9                RET
547                *
548                *
549                *
550 DEB5                   END
```

```
****************************
* S Y M B O L   T A B L E *
****************************

ALNUM   DE09    ALPHA   DE02    BRS10   DDC5    BRSER   DDC0
CINC    DDB4    CMPHL   DE26    COL0    DD55    COMP    DE14
COUTC   DD6A    CRLF    DD5E    DADA    DE30    DADM    DE39
DELAY   DE41    DLY10   DE48    EFC10   DDF4    EFC20   DDFE
```

EFC30	DDFF	EFETCH	DDE0	EXIT1	DD45	FIL10	DE7E
FIL20	DE8D	FILL	DE7C	HLM10	DE96	HLM20	DE9F
HLM90	DEAD	HLM99	DEB1	HLMUL	DE8F	IGNB	DDD2
IGNBR	DDD1	INPL0	DD1A	INPLN	DD1F	INSER	DDB4
IPL10	DD22	IPL20	DD2A	IPL30	DD49	MOV10	DE5F
MOV20	DE6C	MOV30	DE79	MOV40	DE7A	MOVE	DE4F
NUMER	DE0D	OTBIN	DD75	OTC10	DD70	OTC20	DD8E
OTC99	DD8C	OTS10	DD95	OTS20	DD9D	OUTC	DD60
OUTSE	DD94	SCCHR	DD60	SUBDE	DE1A		

```
002                             ORG   :DEB5
003                     *
004                     *
005                     *
006                     *  ==================================
007                     *** BASIC EXECUTION / RUN-TIME MODULE ***
008                     *  ==================================
009                     *
010                     * Generally, BC is used as entry pointer to the
011                     * textbuffer.
012                     *
013                     ********************
014                     * RUN basiccmd NEW *
015                     ********************
016                     *
017                     * Sets up heap, empty textbuffer and symboltable,
018                     * sets pointers of textbuffer and symboltable
019                     * correctly. Old buffer contents is not destroyed,
020                     * (except 4 locations), but not useable.
021                     * Valid as direct command only.
022                     *
023                     * Exit: A=1, CY=1.
024                     *
025 DEB5 00             RNEW    NOP
026 DEB6 00                     NOP
027 DEB7 00                     NOP
028 DEB8 2A9B02                 LHLD  :029B     Get startaddr heap
029 DEBB 229F02                 SHLD  :029F     Set start textbuf=start heap
030 DEBE 3600                   MVI   M,:00     Store 00 in 1st addr.
031 DEC0 23                     INX   H
032 DEC1 22A102                 SHLD  :02A1     Set start symtab
033 DEC4 3600                   MVI   M,:00     00 in 1st addr.
034 DEC6 23                     INX   H
035 DEC7 22A302                 SHLD  :02A3     Set end symtab
036
037                     * Entry from scratch/edit:
038
039 DECA 2A9D02         HRINIT  LHLD  :029D     Get heap size
040 DECD EB                     XCHG            in DE
041 DECE CD95D1                 CALL  :D195     Init heap to all available
042 DED1 3E01                   MVI   A,:01     Code for 'buffer crunched'
043 DED3 37                     STC
044 DED4 C9                     RET
045                     *
046                     ********************
047                     * RUN basiccmd CONT *
048                     ********************
049                     *
050                     * Resets step flag, decr. CONFL (error if it was
051                     * 0), restores FRAME from stack, restores stack-
052                     * pointer and continues program execution.
053                     * Valid as direct command only.
054                     *
055 DED5 AF             RCONT   XRA   A
056 DED6 321601         RCN10   STA   :0116     Reset step flag
057 DED9 212601                 LXI   H,:0126   Get pntr suspended program
058 DEDC 35                     DCR   M         Update it
059 DEDD 3E19                   MVI   A,:19
060 DEDF FAF5D9                 JM    :D9F5     Evt. run error 'CAN'T CONT'
061 DEE2 2A2701                 LHLD  :0127     Get current base stack level
062 DEE5 EB                     XCHG            in DE
063 DEE6 211500                 LXI   H,:0015   FRAME length
```

```
064 DEE9 19                     DAD   D          Top of FRAME in stack
065 DEEA E5                     PUSH  H          Save it
066 DEEB 010001                 LXI   B,:0100    Addr SYSBOT
067 DEEE CD4FDE                 CALL  :DE4F      Load FRAME from stack
068 DEF1 E1                     POP   H          Get addr FRAME top in stack
069 DEF2 222701                 SHLD  :0127      Update current base stack
070                                              level
071 DEF5 F9                     SPHL             Update stackpointer
072 DEF6 2A0201                 LHLD  :0102      Get start current command
073 DEF9 44                     MOV   B,H
074 DEFA 4D                     MOV   C,L        Store it in BC
075 DEFB C37FC8                 JMP   :C87F      Run BASIC line
076                     *
077                     **********************
078                     * RUN basiccmd STEP *
079                     **********************
080                     *
081 DEFE 3EFF           RSTEP   MVI   A,:FF      Init value STEP flag
082 DF00 C3D6DE                JMP   :DED6      Set STEP flag and continu
083                     *
084                     **********************
085                     * RUN basiccmd STOP *
086                     **********************
087                     *
088                     * Entry: No conditions.
089                     * Exit:  A=03, CY=1. HL points after string.
090                     *
091 DF03 CDFFDA         RSTOP   CALL  :DAFF      Print 'STOPPED'
092 DF06 C9DB                   DBL   :DBC9
093 DF08 3E03                  MVI   A,:03      Code for 'susp. execution'
094 DF0A 37                     STC
095 DF0B C9                     RET
096                     *
097                     **********************
098                     * RUN basiccmd END *
099                     **********************
100                     *
101                     * Entry: No conditions.
102                     * Exit:  A=01, CY=1. HL points after string.
103                     *
104 DF0C CDFFDA         REND    CALL  :DAFF      Print 'END PROGRAM'
105 DF0F CFDB                   DBL   :DBCF
106 DF11 3E01                  MVI   A,:01      Code for 'stop execution'
107 DF13 37                     STC
108 DF14 C9                     RET
109                     *
110                     ********************
111                     * RUN basiccmd IF *
112                     ********************
113                     *
114                     * Used for IF .. GOTO <linenr> and for
115                     * IF .. THEN <linenr>.
116                     *
117                     RIFG
118 DF15 CD63E7         RIFTL   CALL  :E763      (0) Run logical expression
119 DF18 3C                     INR   A
120 DF19 CA63DF                 JZ    :DF63      Run GOTO if condition true
121
122                     * If condition false:
123
124 DF1C 03                      INX   B
125 DF1D 03                      INX   B          Skip linenr
```

```
126 DF1E B7                     ORA   A         No special action
127 DF1F C9                     RET
128                     *
129                     ****************************************
130                     * RUN basiccmd IF .. THEN <statement> *
131                     ****************************************
132                     *
133 DF20 CD63E7         RIFTC   CALL  :E763     (O) Run logical expression
134 DF23 3C                     INR   A
135 DF24 C28FE1                 JNZ   :E18F     (O) Ignore rest if false
136
137                     * If condition true:
138
139 DF27 03                     INX   B         Skip length line
140 DF28 B7                     ORA   A         No special action
141 DF29 C9                     RET             Execute rest of line
142                     *
143                     ******************
144                     * basiccmd GOSUB *
145                     ******************
146                     *
147                     * Saves current program state on the interpreter
148                     * stack and branches to a named line. The running
149                     * FOR loop (if any) is saved in order to avoid
150                     * problems if any unpaired NEXT is encountered.
151                     * The stackpointer at the subroutine-entry is held
152                     * to enable breaking out of a FOR-NEXT loop.
153                     *
154 DF2A CDEDE6         RGOSUB  CALL  :E6ED     (O) Get linenr and find it
155
156                     * Entry from ON - GOSUB:
157
158 DF2D D1             RGS10   POP   D         Kill return addr
159 DF2E 221901                 SHLD  :0119     Save new PC
160 DF31 CD3CE1                 CALL  :E13C     (O) Save FOR loop contents
161 DF34 C5                     PUSH  B         Save program position
162 DF35 2A1901                 LHLD  :0119     Get new PC
163 DF38 44                     MOV   B,H       ) Is new text position
164 DF39 4D                     MOV   C,L       )
165 DF3A 2A1301                 LHLD  :0113     Get stack level last GOSUB
166 DF3D E5                     PUSH  H         Save evt link to previous
167                                             subroutine entry
168 DF3E 210000                 LXI   H,:0000
169 DF41 220401                 SHLD  :0104     No running loop
170 DF44 39                     DAD   SP        SP in HL
171 DF45 221301                 SHLD  :0113     SP in STKGOS for return
172
173                     * Entry on return:
174
175 DF48 B7             RGS20   ORA   A         No special action
176 DF49 C38FC8                 JMP   :C88F     Into Basic monitor
177                     *
178                     ************************
179                     * RUN basiccmd RETURN *
180                     ************************
181                     *
182                     * Reverses the effect of a previous 'GOSUB'.
183                     *
184 DF4C D1             RRET    POP   D         Kill returnaddr
185 DF4D 2A1301                 LHLD  :0113     Get stack level last GOSUB
186 DF50 7C                     MOV   A,H
187 DF51 B5                     ORA   L         0 if no active call
```

```
188 DF52 3E01              MVI   A,:01
189 DF54 CAF5D9            JZ    :D9F5      Then run error 'RETURN
190                                         WITHOUT GOSUB'
191 DF57 F9                SPHL             Else: re-instate old stack
192 DF58 E1                POP   H
193 DF59 221301            SHLD  :0113      Link back to previous
194                                         GOSUB
195 DF5C C1                POP   B          Restore text pntr
196 DF5D CD67E1            CALL  :E167      (0) Pop FRAME
197 DF60 C348DF            JMP   :DF48      Back to Basic monitor
198                *
199                **********************
200                * RUN basiccmd GOTO *
201                **********************
202                *
203                * Simply transfers control to a named line.
204                *
205                * Entry from IF - GOTO:
206                *
207 DF63 CDEDE6    RGOTO   CALL  :E6ED      (0) Get linenr and find it
208
209                * Entry from ON - GOTO:
210
211 DF66 44          RGT10   MOV   B,H      ) Set textpointer to line
212 DF67 4D                  MOV   C,L      )
213 DF68 B7                  ORA   A        No special action
214 DF69 C9                  RET
215                *
216                ****************************
217                * RUN basiccmd ON .. GOTO *
218                ****************************
219                *
220 DF6A CD78DF    RONGT   CALL  :DF78      Process command
221 DF6D DA66DF            JC    :DF66      Use GOTO if OK
222 DF70 C9                RET              If outside list
223                *
224                ****************************
225                * RUN basiccmd ON .. GOSUB *
226                ****************************
227                *
228 DF71 CD78DF    RONGS   CALL  :DF78      Process command
229 DF74 DA2DDF            JC    :DF2D      Use GOSUB if OK
230 DF77 C9                RET              If outside list
231                *
232                ************************
233                * COMMON 'ON ..' CODE *
234                ************************
235                *
236                * Exit: CY=1: OK.
237                *       CY=0: Outside list.
238                *
239 DF78 CD1DE7    RONFN   CALL  :E71D      ) (0) Get index of number
240 DF7B 5F                MOV   E,A        ) in list in E
241 DF7C 0A                LDAX  B          Get nr of linenrs in list
242 DF7D 03                INX   B
243 DF7E 6F                MOV   L,A
244 DF7F 2600              MVI   H,:00
245 DF81 29                DAD   H
246 DF82 09                DAD   B          Pointer after list
247 DF83 E5                PUSH  H          Save pointer
248 DF84 1D                DCR   E
249 DF85 1C                INR   E
```

```
250 DF86 CA9BDF             JZ    :DF9B     Index of 0: outside list
251 DF89 BB                 CMP   E
252 DF8A DA9BDF             JC    :DF9B     If index too large
253 DF8D 1600               MVI   D,:00
254 DF8F 1B                 DCX   D
255 DF90 EB                 XCHG
256 DF91 29                 DAD   H
257 DF92 09                 DAD   B         Pntr to reqd linenr
258 DF93 44                 MOV   B,H       ) in BC
259 DF94 4D                 MOV   C,L       )
260 DF95 CDEDE6             CALL  :E6ED     (0) Find reqd line
261 DF98 C1                 POP   B
262 DF99 37                 STC             CY=1: OK
263 DF9A C9                 RET
264
265                 * If outside list:
266
267 DF9B C1         ROF10   POP   B
268 DF9C B7         ROF11   ORA   A         CY=0: outside list
269 DF9D C9                 RET
270                 *
271                 **********************
272                 * RUN basiccmd RUN *
273                 **********************
274                 *
275                 * No linenumber is given.
276                 *
277 DF9E 210000     RRUN    LXI   H,:0000
278 DFA1 221501             SHLD  :0115     Reset trace/step flag
279 DFA4 2A9F02             LHLD  :029F     Get start textbuf
280 DFA7 44         RRN10   MOV   B,H       )
281 DFA8 4D                 MOV   C,L       ) Store it in BC
282 DFA9 CD01E4             CALL  :E401     (0) Run RESTORE; Set data
283                                          pntr to start program
284 DFAC 3100F9             LXI   SP,:F900  Reset stackpointer
285 DFAF AF                 XRA   A
286 DFB0 322601             STA   :0126     No suspended program
287 DFB3 CD23CB             CALL  :CB23     Empty HEAP + symtab
288 DFB6 B7                 ORA   A         No special action
289 DFB7 C38FC8             JMP   :C88F     Run program
290                 *
291                 ******************************
292                 * RUN basiccmd RUN <linenr> *
293                 ******************************
294                 *
295                 * After RUN, a line number is given.
296                 *
297 DFBA CDEDE6     RRUNN   CALL  :E6ED     (0) Read linenr and find
298                                          it in textbuf
299 DFBD C3A7DF             JMP   :DFA7     Process RUN
300                 *
301                 **********************
302                 * RUN basiccmd POKE *
303                 **********************
304                 *
305 DFC0 CDF8E6     RPOKE   CALL  :E6F8     (0) Get addr in HL
306
307                 * Entry for other modules:
308
309 DFC3 CD1DE7     RPEN    CALL  :E71D     (0) Get argument in A
310 DFC6 77                 MOV   M,A       Store it
311 DFC7 B7                 ORA   A         No special action
```

```
312 DFC8 C9                 RET
313                 *
314                 **********************
315                 * RUN basiccmd OUT *
316                 **********************
317                 *
318 DFC9 CD1DE7       ROUT    CALL  :E71D     (0) Get portnr in A
319 DFCC 57                 MOV   D,A       and in D
320 DFCD CD1DE7             CALL  :E71D     (0) Get data in A
321 DFD0 5F                 MOV   E,A       and in E
322 DFD1 B7                 ORA   A
323 DFD2 C3C8D8             JMP   :D8C8     Output to DCE-bus
324                 *
325                 ************************
326                 * RUN basiccmd WAIT *
327                 ************************
328                 *
329                 * WAIT I,J,K reads the status of Real World port
330                 * I, EXOR's it with K and AND's it with J until
331                 * a result equal to J is obtained.
332                 *
333 DFD5 CD1DE7       RWAIT   CALL  :E71D     (0) Get portnr
334 DFD8 57                 MOV   D,A       in D
335 DFD9 CD1DE7             CALL  :E71D     (0) Get bits needed high
336 DFDC 67                 MOV   H,A       in H
337 DFDD 0A                 LDAX  B         Get next byte from text
338 DFDE 03                 INX   B
339 DFDF D6FF               SUI   :FF       Check if ony 2 arguments
340 DFE1 00                 NOP
341 DFE2 C4DACE             CNZ   :CEDA     If 3 arg: Get XOR mask
342 DFE5 6F                 MOV   L,A       in L
343 DFE6 CDE0D8     RWT20   CALL  :D8E0     Input from DCE-bus
344 DFE9 7B                 MOV   A,E       into A
345 DFEA AD                 XRA   L         XOR with mask
346 DFEB A4                 ANA   H         AND with bits needed high
347 DFEC BC                 CMP   H         Correct value reached?
348 DFED C8                 RZ              Then abort
349 DFEE CDA5D6             CALL  :D6A5     Check keyb for new inputs
350 DFF1 D2E6DF             JNC   :DFE6     Next DCE-input if no Break
351
352                 * If suspended:
353
354 DFF4 C312E0             JMP   :E012     (0) Quit: 'cmd broken in'
355                 *
356                 ***************************
357                 * RUN basiccmd WAIT MEM *
358                 ***************************
359                 *
360                 * As WAIT, but with I is a memory location.
361                 *
362 DFF7 CDF8E6       RWTEM   CALL  :E6F8     (0) Get memory addr in HL
363 DFFA CD1DE7             CALL  :E71D     (0) Get bit mask
364 DFFD 57                 MOV   D,A       in D
365 DFFE 0A                 LDAX  B         Get next byte from text
366 DFFF 03                 INX   B
367 E000 D6FF               SUI   :FF       Check if only 2 arguments
368 E002 00                 NOP
369 E003 C4DACE             CNZ   :CEDA     If 3 arg: Get XOR mask
370 E006 5F                 MOV   E,A       in E
371 E007 7B         RWM20   MOV   A,E       XOR mask in A
372 E008 AE                 XRA   M         XOR with memory
373 E009 A2                 ANA   D         AND with bit mask
```

```
374 E00A BA                     CMP   D          Correct value reached?
375 E00B C8                     RZ               Then abort
376 E00C CDA5D6                 CALL  :D6A5      Check keyb for inputs
377 E00F D207E0                 JNC   :E007      Cont if no Break pressed
378
379                     * If suspended:
380
381 E012 3E02           RWT30   MVI   A,:02      Code 'command broken in'
382 E014 37                     STC
383 E015 C9                     RET
384                     *
385                     *
386                     *
387 E016                        END
```

****
* S Y M B O L T A B L E *
****

HRINIT	DECA	RCN10	DED6	RCONT	DED5	REND	DF0C
RGOSUB	DF2A	RGOTO	DF63	RGS10	DF2D	RGS20	DF48
RGT10	DF66	RIFG	DF15	RIFTC	DF20	RIFTL	DF15
RNEW	DEB5	ROF10	DF9B	ROF11	DF9C	RONFN	DF78
RONGS	DF71	RONGT	DF6A	ROUT	DFC9	RPEN	DFC3
RPOKE	DFC0	RRET	DF4C	RRN10	DFA7	RRUN	DF9E
RRUNN	DFBA	RSTEP	DEFE	RSTOP	DF03	RWAIT	DFD5
RWM20	E007	RWT20	DFE6	RWT30	E012	RWTEM	DFF7

```
002                             ORG   :E016
003                     *
004                     *
005                     *
006                     ****************************
007                     * RUN basiccmd WAIT TIME *
008                     ****************************
009                     *
010                     * Timer 01BE/BF is decremented by clock
011                     * interrupt RST7.
012                     *
013 E016 CDF8E6         RWTET   CALL  :E6F8     Get time to wait
014 E019 22BE01                 SHLD  :01BE     Load timer
015 E01C 2ABE01         RWT40   LHLD  :01BE     Get timer
016 E01F 7C                     MOV   A,H
017 E020 B5                     ORA   L
018 E021 C8                     RZ              Abort if (timer)=0
019 E022 CDA5D6                 CALL  :D6A5     Check keyb for new inputs
020 E025 D21CE0                 JNC   :E01C     Again if no break pressed
021
022                     * If suspended:
023
024 E028 C312E0                 JMP   :E012     Abort, 'command broken in'
025                     *
026                     ****************************************
027                     * RUN basiccmd FOR .. TO .. (STEP ..) *
028                     ****************************************
029                     *
030 E02B D1             RFOR    POP   D         Kill return addr
031 E02C CD3CE1                 CALL  :E13C     Save old FRAME on stack
032 E02F CD5AE4                 CALL  :E45A     Init variable
033 E032 220401                 SHLD  :0104     Remember location variable
034 E035 E630                   ANI   :30
035 E037 D610                   SUI   :10       Set flags for var.type
036 E039 CD08E8                 CALL  :E808     Eval TO expr, result in MACC
037 E03C CAA2E0                 JZ    :E0A2     If INT variable
038
039                     * If FPT variable:
040
041 E03F E7                     RST   4         Subtract 'FROM'
042 E040 03                     DATA  :03
043 E041 210601                 LXI   H,:0106   Get addr. LSTPF
044 E044 3600                   MVI   M,:00     Default STEP implicit
045 E046 0A                     LDAX  B         Get evt. STEP val. from text
046 E047 03                     INX   B
047 E048 FEFF                   CPI   :FF
048 E04A CA5FE0                 JZ    :E05F     Jump if no STEP
049
050                     * If STEP:
051
052 E04D CD18C0                 CALL  :C018     Save 'to-from' on stack
053 E050 34                     INR   M         Stepflag explicit
054 E051 0B                     DCX   B
055 E052 CD08E8                 CALL  :E808     Stepvalue in MACC
056 E055 210701                 LXI   H,:0107   Addr. LSTEP
057 E058 E7                     RST   4         Stepvalue in LSTEP
058 E059 0F                     DATA  :0F
059 E05A CD1BC0                 CALL  :C01B     'to-from' range in MACC
060 E05D E7                     RST   4         Find nr of iterations
061 E05E 09                     DATA  :09
062 E05F E7             RFR10   RST   4         Make it INT
063 E060 48                     DATA  :48
```

```
064 E061 210B01      RFR20   LXI   H,:010B   Addr. LCOUNT
065 E064 E7                  RST   4         Iterations in LCOUNT
066 E065 0F                  DATA  :0F
067 E066 7E                  MOV   A,M
068 E067 B7                  ORA   A
069 E068 FC9ECB              CM    :CB9E     Clear LCOUNT if loop in
070                                          wrong direction
071 E06B 60                  MOV   H,B       ) Current pos in start of
072 E06C 69                  MOV   L,C       ) loop in HL
073 E06D 220F01              SHLD  :010F     Set pointer to start loop
074
075                  * Now delete any previous use of the loop:
076
077 E070 011000              LXI   B,:0010   Size of 1 'FOR' stackframe
078 E073 2A0401              LHLD  :0104     Get current loop variable
079 E076 EB                  XCHG            in DE
080 E077 210000              LXI   H,:0000
081 E07A 39                  DAD   SP        Stack start
082 E07B C37FE0              JMP   :E07F     Into loop
083
084                  * Loop:
085
086 E07E 09          RFR30   DAD   B         Up 1 frame
087 E07F 7E          RFR40   MOV   A,M
088 E080 23                  INX   H
089 E081 B6                  ORA   M
090 E082 CA9FE0              JZ    :E09F     Jump if top of stack
091 E085 7E                  MOV   A,M
092 E086 2B                  DCX   H
093 E087 BA                  CMP   D         Comp top byte variable addr
094 E088 C27EE0              JNZ   :E07E     Again if not the same
095 E08B 7E                  MOV   A,M
096 E08C 93                  SUB   E
097 E08D C27EE0              JNZ   :E07E     Cont if bottombyte different
098 E090 E5                  PUSH  H         Frame bottom to be removed
099 E091 210200              LXI   H,:0002
100 E094 39                  DAD   SP        Update stackpointer
101 E095 54                  MOV   D,H       ) DE is bottom of area to be
102 E096 5D                  MOV   E,L       ) moved
103 E097 09                  DAD   B         Add 1 frame
104 E098 44                  MOV   B,H       ) Place to move area to
105 E099 4D                  MOV   C,L       )
106 E09A E1                  POP   H         Top area to move
107 E09B CD4FDE              CALL  :DE4F     Remove old frame
108 E09E F9                  SPHL            New stack position
109 E09F C314E1      RFR50   JMP   :E114     Use common mode to re-
110                                          instate textpointer
111
112                  * If INT variable:
113
114 E0A2 E7          RFR70   RST   4         Subtract 'from'
115 E0A3 51                  DATA  :51
116 E0A4 210601              LXI   H,:0106   Get addr. LSTPF
117 E0A7 3680                MVI   M,:80     Default step implicit
118 E0A9 0A                  LDAX  B         Get evt STEP value
119 E0AA 03                  INX   B
120 E0AB FEFF                CPI   :FF
121 E0AD CA61E0              JZ    :E061     If no step given
122
123                  * If STEP:
124
125 E0B0 CD18C0              CALL  :C018     Save 'to-from' on stack
```

```
126 E0B3 34                 INR   M         Stepflag explicit
127 E0B4 0B                 DCX   B
128 E0B5 CD08E8             CALL  :E808     Get stepvalue in MACC
129 E0B8 210701             LXI   H,:0107   Addr. stepvalue if explicit
130 E0BB E7                 RST   4         Stepvalue in LSTEP
131 E0BC 0F                 DATA  :0F
132 E0BD CD1BC0             CALL  :C01B     'to-from' range in MACC
133 E0C0 E7                 RST   4         Find nr of iterations
134 E0C1 57                 DATA  :57
135 E0C2 C361E0             JMP   :E061     Handle loop
136                 *
137                 ****************************************
138                 * RUN basiccmd NEXT <named variable> *
139                 ****************************************
140                 *
141 E0C5 D1         RNEXI  POP   D          Returnaddr
142 E0C6 CD63E9             CALL  :E963     Get varptr in HL
143 E0C9 EB         RNX10  XCHG
144 E0CA 2A0401             LHLD  :0104     Get current loop variable
145 E0CD 7D                 MOV   A,L
146 E0CE B4                 ORA   H         Loopvariable 0?
147 E0CF CA2EDA             JZ    :DA2E     Then run error 'NEXT
148                                           WITHOUT FOR'
149 E0D2 CD14DE             CALL  :DE14     Compare loop and named
150                                           variable pntrs
151 E0D5 CAEEE0             JZ    :E0EE     Perform NEXT if identical
152 E0D8 EB                 XCHG
153 E0D9 221901             SHLD  :0119     Store in scratch area
154 E0DC CD67E1             CALL  :E167     Re-instate next loopvariable
155 E0DF 2A1901             LHLD  :0119     Get it back
156 E0E2 C3C9E0             JMP   :E0C9     Try for a match
157                 *
158                 **********************
159                 * RUN basiccmd NEXT *
160                 **********************
161                 *
162                 * No variable name is given.
163                 *
164 E0E5 D1         RNEXT  POP   D          Returnaddr
165 E0E6 2A0401             LHLD  :0104     Get current loop variable
166 E0E9 7D                 MOV   A,L
167 E0EA B4                 ORA   H         Loopvar is 0?
168 E0EB CA2EDA             JZ    :DA2E     Then run error 'NEXT
169                                           WITHOUT FOR'
170 E0EE 3A0601     RNX20  LDA   :0106     Get LSTPF
171 E0F1 B7                 ORA   A
172 E0F2 FA33E1             JM    :E133     Jump if INT loop variable
173
174                 * If FPT loop variable:
175
176 E0F5 1F                 RAR
177 E0F6 DA1DE1             JC    :E11D     Jump if explicit step
178 E0F9 CD06C0             CALL  :C006     Incr. variable in memory
179 E0FC 210E01     RNX30  LXI   H,:010E   Addr lobyte LCOUNT
180 E0FF 7E         RNX40  MOV   A,M        Get lobyte
181 E100 D601               SUI   :01
182 E102 77                 MOV   M,A       Decr it
183 E103 D214E1             JNC   :E114     Continu if no overflow
184 E106 2B                 DCX   H         Pnts to next byte LCOUNT
185 E107 7D                 MOV   A,L
186 E108 FE0A               CPI   :0A       Hibyte done?
187 E10A C2FFE0             JNZ   :E0FF     More bytes if not
```

```
188
189                     * Loop finished:
190
191 E10D CD67E1              CALL  :E167     Pop frame
192 E110 B7                 ORA   A         No special action
193 E111 C38FC8             JMP   :C88F     Exit to Basic monitor
194
195                     * More time round (entry from 'FOR'):
196
197 E114 2A0F01      RNX50   LHLD  :010F     Get pntr to start loop
198 E117 44                 MOV   B,H       ) in BC
199 E118 4D                 MOV   C,L       )
200 E119 B7                 ORA   A         No special action
201 E11A C38FC8             JMP   :C88F     Exit to Basic monitor
202
203                     * Explicit step:
204
205 E11D E7          RNX60   RST   4         Get value loopvar in MACC
206 E11E 0C                 DATA  :0C
207 E11F E5                 PUSH  H
208 E120 210701             LXI   H,:0107   Addr. LSTEP
209 E123 D228E1             JNC   :E128     If INT
210 E126 E7                 RST   4         FPT: add stepvalue
211 E127 00                 DATA  :00
212 E128 DA2DE1      LOE19   JC    :E12D     If FPT
213 E12B E7                 RST   4         INT: add stepvalue
214 E12C 4E                 DATA  :4E
215 E12D E1          LOE20   POP   H
216 E12E E7                 RST   4         Store new value in
217 E12F 0F                 DATA  :0F       variable
218 E130 C3FCE0             JMP   :E0FC     Test end of loop
219
220                     * If INT loopvariable:
221
222 E133 EA1DE1      RNX70   JPE   :E11D     Jump if explicit step
223 E136 CD0FC0             CALL  :C00F     Incr. variable in memory
224 E139 C3FCE0             JMP   :E0FC     Test end of loop
225                     *
226                     **************
227                     * PUSH FRAME *
228                     **************
229                     *
230                     * Several pointers are save on stack during
231                     * execution of FOR-NEXT loops.
232                     *
233 E13C D1          PUSHF   POP   D         Get addr. where to continue
234 E13D 2A0401             LHLD  :0104     Get current loop variable
235 E140 7C                 MOV   A,H
236 E141 B5                 ORA   L         Check if 0
237 E142 CA64E1             JZ    :E164     Then abort routine
238 E145 2A0F01             LHLD  :010F
239 E148 E5                 PUSH  H         Save pntr to start loop
240 E149 2A1101             LHLD  :0111
241 E14C E5                 PUSH  H         Save pntr to start loop line
242 E14D 2A0B01             LHLD  :010B
243 E150 E5                 PUSH  H         ) Save loop iteration count
244 E151 2A0D01             LHLD  :010D     ) (4 bytes)
245 E154 E5                 PUSH  H         )
246 E155 2A0701             LHLD  :0107
247 E158 E5                 PUSH  H         ) Save step value
248 E159 2A0901             LHLD  :0109     ) (4 bytes)
249 E15C E5                 PUSH  H         )
```

```
250 E15D 3A0601             LDA   :0106
251 E160 F5                 PUSH  PSW       Save LSTPF
252 E161 2A0401             LHLD  :0104
253 E164 E5         PU1     PUSH  H         Save current loop variable
254 E165 EB                 XCHG            Addr. to continue in HL
255 E166 E9                 PCHL            Set program counter
256                 *
257                 *************
258                 * POP FRAME *
259                 *************
260                 *
261                 * Restores looppointers in RAM.
262                 *
263 E167 D1         POPF    POP   D
264 E168 E1                 POP   H
265 E169 220401             SHLD  :0104     Restore LOPVAR
266 E16C 7C                 MOV   A,H
267 E16D B5                 ORA   L         LOPVAR=0?
268 E16E CA8DE1             JZ    :E18D     Then abort routine
269 E171 F1                 POP   PSW
270 E172 320601             STA   :0106     Restore LSTPF
271 E175 E1                 POP   H         )
272 E176 220901             SHLD  :0109     ) Restore LSTEP
273 E179 E1                 POP   H         ) (4 bytes)
274 E17A 220701             SHLD  :0107
275 E17D E1                 POP   H         )
276 E17E 220D01             SHLD  :010D     ) Restore LCOUNT
277 E181 E1                 POP   H         ) (4 bytes)
278 E182 220B01             SHLD  :010B
279 E185 E1                 POP   H
280 E186 221101             SHLD  :0111     Restore LOPLN
281 E189 E1                 POP   H
282 E18A 220F01             SHLD  :010F     Restore LOPPT
283 E18D D5         PP1     PUSH  D
284 E18E C9                 RET
285                 *
286                 ************************************
287                 * RUN basiccmds DATA - REM - IMP *
288                 ************************************
289                 *
290                 * RREM/RDATA:
291                 *    Entry: B: Points to length byte of string.
292                 *    Exit:  DEHL preserved. AF corrupted.
293                 * RIMP:
294                 *    No action.
295                 *
296                 RREM
297 E18F 0A        RDATA   LDAX  B         Get length of string
298
299                 * Entry for REXPS:
300
301 E190 03        RRM10   INX   B         BC points to 1st char
302 E191 81                 ADD   C
303 E192 4F                 MOV   C,A       BC points to end of string
304 E193 D0                 RNC
305 E194 04                 INR   B         If overflow: correct B
306
307                 * Entry for RUN IMP:
308
309 E195 B7        RIMP    ORA   A         No special action
310 E196 C9                 RET
```

```
                        **********************
                        * RUN basiccmd LIST *
                        **********************
                        *
                        * The whole textbuffer contents is listed.
                        *
                        RLIST
E197 3EFF               RLIS0   MVI   A,:FF     Init. mode 0
E199 EF                         RST   5         Change mode
E19A 18                         DATA  :18
E19B 3E0C                       MVI   A,:0C
E19D EF                         RST   5         Clear screen
E19E 03                         DATA  :03
E19F 2A9F02             RLS05   LHLD  :029F     Get startaddr. textbuf
E1A2 EB                         XCHG            in DE
E1A3 2AA102                     LHLD  :02A1     Get start symtab
E1A6 2B                         DCX   H         End textbuf in HL
E1A7 C3CFE1                     JMP   :E1CF     Perform listing
                        *
                        ************************************
                        * RUN basiccmd LIST <linenumber> *
                        ************************************
                        *
                        * Entry: BC points to linenumber.
                        * Exit:  BC updated. AFDEHL corrupted.
                        *
E1AA CDE7E6             RLIS1   CALL  :E6E7     Read linenr and find
                                                it in textbuf
E1AD 00                         NOP
E1AE 54                         MOV   D,H       ) Linenr in DE
E1AF 5D                         MOV   E,L       )
E1B0 DC39DE                     CC    :DE39     If linenr found: calc addr
                                                after string in HL
E1B3 C3CFE1                     JMP   :E1CF     Perform listing
                        *
                        ******************************
                        * RUN basiccmd LIST <range> *
                        ******************************
                        *
                        * Entry: BC points to 1st linenumber.
                        * Exit:  BC updated. AFDEHL corrupted.
                        *
E1B6 2A9F02             RLIS2   LHLD  :029F     Get start textbuf
E1B9 CDD9E6                     CALL  :E6D9     Read 1st linenr
E1BC C4F6CA                     CNZ   :CAF6     If given: Find it in textbuf
E1BF EB                         XCHG            Addr in DE
E1C0 2AA102                     LHLD  :02A1     Get start symtab
E1C3 2B                         DCX   H         End textbuf in HL
E1C4 CDD9E6                     CALL  :E6D9     Read 1st linenr
E1C7 37                         STC
E1C8 3F                         CMC
E1C9 C4F6CA                     CNZ   :CAF6     If 1st nr found: find 2nd
E1CC DC39DE                     CC    :DE39     If found: Calc addr after
                                                string in HL


                        * Perform listing:

E1CF C5                 RLS10   PUSH  B
E1D0 42                         MOV   B,D       ) Start listed area in BC
E1D1 4B                         MOV   C,E       )
E1D2 221B01                     SHLD  :011B     Store end listed area
E1D5 EB                         XCHG            also in DE
```

```
374 E1D6 221901              SHLD  :0119     Store start listed area
375 E1D9 60          RLS20   MOV   H,B       ) Startaddr in HL
376 E1DA 69                  MOV   L,C       )
377 E1DB CD14DE              CALL  :DE14     Check if all lines listed
378 E1DE CAF2E1              JZ    :E1F2     Abort if ready
379 E1E1 CD73D8              CALL  :D873     List curent line if linenr
380                                          correct
381 E1E4 CDBBD6              CALL  :D6BB     Scan keyboard
382 E1E7 DAF2E1              JC    :E1F2     Break pressed: stop listing
383 E1EA 00                  NOP
384 E1EB 00                  NOP
385 E1EC C4DAD6              CNZ   :D6DA     If a key is pressed:
386                                          Wait for spacebar
387 E1EF D2D9E1              JNC   :E1D9     No break: list further
388
389                 * If ready:
390
391 E1F2 B7          RLS30   ORA   A         No special action
392 E1F3 C1                  POP   B
393 E1F4 C9                  RET
394                 *
395                 *
396                 *
397 E1F5                     END
```

```
*****************************
* S Y M B O L   T A B L E *
*****************************
```

L0E19	E128	L0E20	E12D	POPF	E167	PP1	E18D
PU1	E164	PUSHF	E13C	RDATA	E18F	RFOR	E02B
RFR10	E05F	RFR20	E061	RFR30	E07E	RFR40	E07F
RFR50	E09F	RFR70	E0A2	RIMP	E195	RLIS0	E197
RLIS1	E1AA	RLIS2	E1B6	RLIST	E197	RLS05	E19F
RLS10	E1CF	RLS20	E1D9	RLS30	E1F2	RNEXI	E0C5
RNEXT	E0E5	RNX10	E0C9	RNX20	E0EE	RNX30	E0FC
RNX40	E0FF	RNX50	E114	RNX60	E11D	RNX70	E133
RREM	E18F	RRM10	E190	RWT40	E01C	RWTET	E016

```
002                             ORG   :E1F5
003                     *
004                     *
005                     *
006                     **********************
007                     * RUN basiccmd EDIT *
008                     **********************
009                     *
010                     * The editbuffer is set up by moving the program
011                     * to the end of the free RAM space. All memory
012                     * between heapstart and textbegin is used as
013                     * editbuffer.
014                     * On break + space: The edited area is deleted
015                     * from the textbuffer and the program is moved
016                     * to just after the end of the edited text by
017                     * changing the heapsize.
018                     * EFSW is set for input from editbuffer.
019                     *
020 E1F5 CD65E2         REDIT   CALL  :E265     Init. edit buffer
021 E1F8 CD9FE1                 CALL  :E19F     List into edit buffer
022 E1FB AF             RED05   XRA   A
023 E1FC 323101                 STA   :0131     Set output to screen
024 E1FF 32B902                 STA   :02B9     Enable complete keyb.scan
025 E202 CD75DD                 CALL  :DD75     0 on end of buffer
026 E205 210000                 LXI   H,:0000
027 E208 22B400                 SHLD  :00B4     Clear tab table pntr
028 E20B EF                     RST   5         Init screen editor
029 E20C 2A                     DATA  :2A
030 E20D CDBED6         RED10   CALL  :D6BE     Get char from keyboard
031 E210 DA1BE2                 JC    :E21B     If break pressed
032 E213 CA0DE2                 JZ    :E20D     Wait for inputs
033 E216 EF                     RST   5         Obey character
034 E217 2D                     DATA  :2D
035 E218 C30DE2                 JMP   :E20D     Get next input
036
037                     * If Break pressed:
038
039 E21B 3E0C           RED19   MVI   A,:0C
040 E21D EF                     RST   5         Clear screen
041 E21E 03                     DATA  :03
042 E21F CDBED6         RED20   CALL  :D6BE     Get char from keyboard
043 E222 DA4DE2                 JC    :E24D     If again Break
044
045                     * Break followed by any character:
046
047 E225 CA1FE2                 JZ    :E21F     Wait for a char typed in
048 E228 3E02                   MVI   A,:02
049 E22A 323501                 STA   :0135     EFSW: input from buffer
050 E22D 2A1B01                 LHLD  :011B     Get end listed area
051 E230 EB                     XCHG            in DE
052 E231 2A1901                 LHLD  :0119     Get start listed area
053 E234 CD1ADE                 CALL  :DE1A     Calc. length listed area
054 E237 00                     NOP
055 E238 CDD1C9                 CALL  :C9D1     Delete edited area in txtbuf
056 E23B 2A9B02                 LHLD  :029B     Get start HEAP
057 E23E EB                     XCHG            in DE
058 E23F 2AA400                 LHLD  :00A4     Get input pntr editbuf
059 E242 CD1ADE                 CALL  :DE1A     Calc length used edit area
060 E245 23                     INX   H
061 E246 23                     INX   H
062 E247 EB                     XCHG            DE: length edit area +2
063 E248 CD95D1                 CALL  :D195     Program to end of editbuf
```

```
064 E24B B7                 ORA   A          No special conditions
065 E24C C9                 RET
066
067                 * Break followed by 2nd break:
068
069 E24D CDCADE     RED30   CALL  :DECA      Restore original Heap +
070                                          program buffers
071 E250 B7                 ORA   A          No special conditions
072 E251 C9                 RET
073 E252 00                 NOP
074                 *
075                 ********************************
076                 * RUN basiccmd EDIT <linenr> *
077                 ********************************
078                 *
079 E253 CD65E2     REDI1   CALL  :E265      Init edit buffer
080 E256 CDAAE1             CALL  :E1AA      List one line
081 E259 C3FBE1             JMP   :E1FB      Into Run edit
082                 *
083                 ********************************
084                 * RUN basiccmd EDIT <range> *
085                 ********************************
086                 *
087 E25C CD65E2     REDI2   CALL  :E265      Init edit buffer
088 E25F CDB6E1             CALL  :E1B6      List part of program
089 E262 C3FBE1             JMP   :E1FB      Into Run edit
090                 *
091                 ******************************
092                 * INITIALISE SCREEN EDITOR *
093                 ******************************
094                 *
095                 * Sets up a mode 0 screen, clears all variables
096                 * and arrays. Moves Basic program to top of free
097                 * memory, initialises edit pointers.
098                 *
099                 * Exit: BC preserved. AFDEHL corrupted.
100                 *
101 E265 3EFF       REDIN   MVI   A,:FF
102 E267 EF                 RST   5          Change screen to mode 0
103 E268 18                 DATA  :18
104 E269 CD6DD8             CALL  :D86D      Run 'OUT OF MEMORY' error
105                                          if insufficient space. Else
106                                          empty HEAP + variables
107 E26C CD51EB             CALL  :EB51      Calc free RAM space
108 E26F EB                 XCHG             in DE
109 E270 2A9D02             LHLD  :029D      Get HEAPsize
110 E273 19                 DAD   D
111 E274 EB                 XCHG             Total 'free' RAM in DE
112 E275 CD95D1             CALL  :D195      Program to end free RAM
113 E278 2A9F02             LHLD  :029F      Get startaddr. textbuf
114 E27B 2B                 DCX   H
115 E27C 2B                 DCX   H          Minus 2
116 E27D 22A600             SHLD  :00A6      Store end available space
117 E280 2A9B02             LHLD  :029B      Get startaddr HEAP
118 E283 23                 INX   H
119 E284 23                 INX   H          Plus 2
120 E285 22A200             SHLD  :00A2      Store startaddr. editbuf
121 E288 22A400             SHLD  :00A4      Set input pntr editbuf
122 E28B 213101             LXI   H,:131
123 E28E 3602               MVI   M,:02      Set output to editbuf
124 E290 C9                 RET
```

```
                        ****************************
                        * INPUT FROM EDIT BUFFER *
                        ****************************
                        *
                        * Entry: A=0, CY=0.
                        *
E291 F5         IFBNL   PUSH  PSW
E292 2AA200             LHLD  :00A2     Get startaddr. editbuf
E295 223201             SHLD  :0132     Store it in EFEPT
E298 7E         IFB10   MOV   A,M       Get char from editbuf
E299 B7                 ORA   A         Char is 0?
E29A CAAAE2             JZ    :E2AA     Then editbuf empty
E29D 23                 INX   H
E29E FE0D               CPI   :0D       Car.ret?
E2A0 C298E2             JNZ   :E298     Get next char if not
                        * If char is car.ret:
E2A3 22A200             SHLD  :00A2     Update startaddr. editbuf
E2A6 0E00               MVI   C,:00     1st pos on line
E2A8 F1                 POP   PSW       No special action
E2A9 C9                 RET
                        * If buffer empty:
E2AA 323501     IFB20   STA   :0135     Set input from keyboard
E2AD CDCADE             CALL  :DECA     Organise HEAP + buffers
E2B0 F1                 POP   PSW       special action
E2B1 37                 STC             CY=1
E2B2 C9                 RET
                        *
                        ************************
                        * RUN basiccmd PRINT *
                        ************************
                        *
                        * Entry: BC: Position in textbuffer.
                        *
E2B3 0A         RPRINT  LDAX  B         Get length
E2B4 03                 INX   B
E2B5 57                 MOV   D,A       Count of expr in D
E2B6 B7                 ORA   A
E2B7 CAF7E2             JZ    :E2F7     Jump if only car.ret
E2BA D5         RPR10   PUSH  D         Save nr of expressions
E2BB 0A                 LDAX  B         Get expr type
E2BC 03                 INX   B
E2BD FE20               CPI   :20
E2BF CAD2E2             JZ    :E2D2     Jump if string
                        * If INT/FPT number:
E2C2 FE00               CPI   :00
E2C4 CD08E8             CALL  :E808     Eval expr. Result in MACC
E2C7 F5                 PUSH  PSW       Save flags on expr.type
E2C8 C453DB             CNZ   :DB53     If INT: print INT number
E2CB F1                 POP   PSW
E2CC CC59DB             CZ    :DB59     If FPT: print FPT number
E2CF C3E3E2             JMP   :E2E3
                        * If string:
E2D2 CD91E7     RPR40   CALL  :E791     Evaluate string expr
E2D5 E5                 PUSH  H
```

```
188 E2D6 EF                     RST   5         Ask cursor pos and size
189 E2D7 0C                     DATA  :0C       char screen
190 E2D8 7B                     MOV   A,E       X-size of screen in A
191 E2D9 95                     SUB   L         Minus X-coord cursor pos
192 E2DA 3C                     INR   A         +1
193 E2DB E1                     POP   H
194 E2DC BE                     CMP   M         (not used further: off
195 E2DD 00                     NOP             screen printing possible)
196 E2DE 00                     NOP             (should be:  CC :DD5E)
197 E2DF 00                     NOP
198 E2E0 CD32DB                 CALL  :DB32     Print string pntd by HL
199                     *
200 E2E3 0A             RPR50   LDAX  B         Get byte after string
201 E2E4 03                     INX   B
202 E2E5 FEFF                   CPI   :FF       End marker?
203 E2E7 CAF6E2                 JZ    :E2F6     Then quit with car.ret
204 E2EA FE3B                   CPI   :3B       ';'?
205 E2EC C40DDB                 CNZ   :DB0D     Cursor to next column if
206                                             not (must be ',')
207 E2EF D1                     POP   D         Get nr of expr to print
208 E2F0 15                     DCR   D         Update expr count
209 E2F1 C2BAE2                 JNZ   :E2BA     Loop if more expressions
210 E2F4 B7                     ORA   A         No special action
211 E2F5 C9                     RET
212
213                     * If end of print statement:
214
215 E2F6 D1             RPR70   POP   D
216 E2F7 CD5EDD         RPR80   CALL  :DD5E     Print 'CR'
217 E2FA B7                     ORA   A
218 E2FB C9                     RET
219                     *
220                     ************************
221                     * RUN basiccmd INPUT *
222                     ************************
223                     *
224                     * Runs an input statement with a prompt ('?').
225                     * RINPQ:  Input with a string.
226                     * RINPUT: Input without a string.
227                     *
228 E2FC CD91E7         RINPQ   CALL  :E791     Evaluate stringexpression
229 E2FF CD32DB                 CALL  :DB32     Print string pntd by HL
230 E302 210200         RINPUT  LXI   H,:0002
231 E305 39                     DAD   SP
232 E306 CD47E4                 CALL  :E447     Update ERSSP, print '?' and
233                                             ask for inputs
234 E309 D464E3                 CNC   :E364     If no break: store inputs
235 E30C F5                     PUSH  PSW       Save last input
236 E30D AF                     XRA   A
237 E30E 321701                 STA   :0117     Reset flag running inputs
238 E311 F1                     POP   PSW       Get last input
239 E312 DA20E3                 JC    :E320     Abort if break
240 E315 1C                     INR   E
241 E316 CA1EE3                 JZ    :E31E     Quit if correct nr of inputs
242 E319 CDFFDA                 CALL  :DAFF     Else: Print
243 E31C 08D2                   DBL   :D208     'SOME INPUT IGNORED'
244 E31E B7             RIP10   ORA   A         No special action
245 E31F C9                     RET
246
247                     * If suspended:
248
249 E320 3E02           RIP20   MVI   A,:02     Code 'cmd broken in'
```

```
250 E322 C9                 RET
251                 *
252                 **********************
253                 * RUN basiccmd READ *
254                 **********************
255                 *
256 E323 3E01       RREAD   MVI   A,:01
257 E325 323501             STA   :0135     Set input from string
258 E328 3A2301             LDA   :0123     Get offset next char to en-
259 E32B 5F                 MOV   E,A       code and store it in E
260 E32C 2A9102             LHLD  :0291     Get DATAQ addr
261 E32F 7E                 MOV   A,M       Get length of string
262 E330 323401             STA   :0134     Store it in EFECT
263 E333 23                 INX   H         Pnts to 1st char
264 E334 223201             SHLD  :0132     Addr 1st char in EFEPT
265 E337 CD64E3             CALL  :E364     Store data
266 E33A 7B                 MOV   A,E       Get offset next char
267 E33B 322301             STA   :0123     Store it
268 E33E 2A3201             LHLD  :0132     Get EFEPT
269 E341 2B                 DCX   H         Pnts to next dataline
270 E342 229102             SHLD  :0291     Store it in DATAQ
271 E345 AF                 XRA   A
272 E346 323501             STA   :0135     Set input from keyboard
273 E349 C9                 RET
274                 *
275                 **********************
276                 * DATA - (not used) *
277                 **********************
278                 *
279 E34A 21         LOE351  DATA  :21
280 E34B 06                 DATA  :06
281 E34C 60                 DATA  :60
282 E34D 69                 DATA  :69
283 E34E 22                 DATA  :22
284 E34F 1F                 DATA  :1F
285                 *
286                 ******************
287                 * RESTART INPUT *
288                 ******************
289                 *
290                 * Error handling if running inputs.
291                 *
292 E350 2A1D01      INPRS   LHLD  :011D     Get saved stackpntr
293 E353 F9                 SPHL            Reload stackpntr
294 E354 2A0201             LHLD  :0102     Get start current command
295 E357 44                 MOV   B,H       ) Store it in BC
296 E358 4D                 MOV   C,L       )
297 E359 CDFFDA             CALL  :DAFF     Print 'RETYPE LINE'
298 E35C 93DB               DBL   :DB93
299 E35E C37FC8             JMP   :C87F     Re-execute input statement
300                 *
301                 ************************************
302                 * STORE DATA IN CORRECT LOCATION *
303                 ************************************
304                 *
305                 * Entry LOE56 not used.
306                 *
307                 * Stores data read from datastatements or gotten
308                 * from an input line on the correct location.
309                 *
310                 * Entry: E : Offset next character to encode
311                 *            (#0291).
```

```
312                     *       HL: Startaddress data line.
313                     *
314 E361 CD36E4         LOE56   CALL  :E436     (not used)
315                     *
316 E364 0A             RRDIP   LDAX  B         Get nr of data reqd
317 E365 03                     INX   B
318 E366 57                     MOV   D,A       Into D
319 E367 15             RRI10   DCR   D
320 E368 FAC1E3                 JM    :E3C1     Jump if ready
321 E36B 1C             RRI20   INR   E         Offset nex char
322 E36C CAA8E3                 JZ    :E3A8     Jump if at end of line
323 E36F 1D                     DCR   E
324 E370 CD63E9         RRI30   CALL  :E963     Get varptr in HL
325 E373 C5                     PUSH  B
326 E374 D5                     PUSH  D
327 E375 4B                     MOV   C,E       Offset in C
328 E376 E5                     PUSH  H         Save varptr
329 E377 213E01                 LXI   H,:013E   Startaddr EBUF
330 E37A E5                     PUSH  H         Save it on stack
331 E37B E630                   ANI   :30
332 E37D CF                     RST   1         Encode a constant
333 E37E 06                     DATA  :06
334 E37F 59                     MOV   E,C       Offset in E
335 E380 C1                     POP   B
336 E381 E1                     POP   H         Get varptr
337 E382 FE20                   CPI   :20       String type ?
338 E384 7B                     MOV   A,E       Offset in A
339 E385 CAA2E3                 JZ    :E3A2     Jump if string type
340 E388 CDB4E4                 CALL  :E4B4     Perform LET (INT/FPT)
341
342                     * Check separator/terminator:
343
344 E38B 4F             RRI40   MOV   C,A       Offset in C
345 E38C CDD2DD                 CALL  :DDD2     Get char from line, neglect
346                                             tab and space
347 E38F 0C                     INR   C         Offset +1
348 E390 FE2C                   CPI   :2C       ',' ?
349 E392 CA99E3                 JZ    :E399
350 E395 FE0D                   CPI   :0D       CR ?
351 E397 0EFF                   MVI   C,:FF     Dummy offset for end of
352                                             line
353 E399 D1             RRI50   POP   D
354 E39A 59                     MOV   E,C       Offset in E
355 E39B C1                     POP   B
356 E39C C2C3E3                 JNZ   :E3C3     Error if char not ',' or
357                                             car.ret
358 E39F C367E3                 JMP   :E367     Read next data line
359
360                     * If string type:
361
362 E3A2 CDB8E4           RRI60   CALL  :E4B8     Perform LET (STR)
363 E3A5 C38BE3                   JMP   :E38B     Check terminator/separator
364
365                     * If end of line reached ('CR'):
366
367 E3A8 3A1701          RRI70   LDA   :0117     Get flag for running inputs
368 E3AB B7                     ORA   A
369 E3AC CABBE3                 JZ    :E3BB     Quit if not running inputs
370 E3AF CD55DD                 CALL  :DD55     Cursor to begin next line
371 E3B2 CDD0E3                 CALL  :E3D0     Print '?', read input line
372 E3B5 DAC2E3                 JC    :E3C2     Abort if break pressed
                                JMP   :E370     Cont reading
```

```
374
375                     * If whole line read:
376
377 E3BB CDDCE3       RRI80   CALL  :E3DC      Find next dataline in txtbuf
378 E3BE C370E3               JMP   :E370      Cont reading
379
380                     * If reading done:
381
382 E3C1 B7           RRI90   ORA   A          No special action
383 E3C2 C9           RRI99   RET
384
385                     * If error:
386
387 E3C3 2A3201       RRISN   LHLD  :0132      Get EFEPT
388 E3C6 11FCFF             LXI   D,:FFFC
389 E3C9 19                 DAD   D            EFEPT-4
390 E3CA 220001             SHLD  :0100        Store start current line
391 E3CD C30BDA             JMP   :DA0B        Run 'SYNTAX ERROR'
392                     *
393                     ****************************
394                     * PREPARE GETTING INPUTS *
395                     ****************************
396                     *
397                     * Continuation of 0E447.
398                     * Prints a prompt ('?') on the screen and gets a
399                     * textline from input.
400                     *
401 E3D0 D5           INPGT   PUSH  D
402 E3D1 EF                   RST   5            Ask cursor pos. and size
403 E3D2 0C                   DATA  :0C          char. screen
404 E3D3 D1                   POP   D
405 E3D4 3E3F                 MVI   A,:3F
406 E3D6 CDA0C6               CALL  :C6A0        Print '?'; input a textline
407 E3D9 5D                   MOV   E,L
408 E3DA 1C                   INR   E            E pnts after last char of
409                                               input
410 E3DB C9                   RET
411                     *
412                     ************************************
413                     * FIND NEXT DATALINE IN TEXTBUFFER *
414                     ************************************
415                     *
416 E3DC F5           DATAF   PUSH  PSW
417 E3DD D5                   PUSH  D
418 E3DE E5                   PUSH  H
419 E3DF 2A2401               LHLD  :0124        Get addr current dataline
420 E3E2 7E           DTF10   MOV   A,M          Get length data string
421 E3E3 B7                   ORA   A
422 E3E4 3E02                 MVI   A,:02
423 E3E6 CAF5D9               JZ    :D9F5        Run error 'OUT OF DATA' if
424                                               no data available
425 E3E9 54                   MOV   D,H          ) Addr dataline in DE
426 E3EA 5D                   MOV   E,L          )
427 E3EB CD39DE               CALL  :DE39        Calc end dataline
428 E3EE 222401               SHLD  :0124        Store it in DATAP
429 E3F1 EB                   XCHG               End in DE, start in HL
430 E3F2 23                   INX   H
431 E3F3 23                   INX   H
432 E3F4 23                   INX   H            Pnts to token
433 E3F5 7E                   MOV   A,M          Get token
434 E3F6 FEA2                 CPI   :A2          Is it DATA?
435 E3F8 EB                   XCHG
```

```
436 E3F9 C2E2E3              JNZ   :E3E2     Check next textline if not
437 E3FC EB                  XCHG
438 E3FD C336E4              JMP   :E436     Continu
439                    *
440 E400 FF                  DATA  :FF
441                    *
442                    *
443                    *
444 E401                     END

****************************
* S Y M B O L   T A B L E *
****************************

DATAF  E3DC   DTF10  E3E2   IFB10  E298   IFB20  E2AA
IFBNL  E291   INPGT  E3D0   INPRS  E350   LOE351 E34A
LOE56  E361   RED05  E1FB   RED10  E20D   RED19  E21B
RED20  E21F   RED30  E24D   REDI1  E253   REDI2  E25C
REDIN  E265   REDIT  E1F5   RINPQ  E2FC   RINPUT E302
RIP10  E31E   RIP20  E320   RPR10  E2BA   RPR40  E2D2
RPR50  E2E3   RPR70  E2F6   RPR80  E2F7   RPRINT E2B3
RRDIP  E364   RREAD  E323   RRI10  E367   RRI20  E36B
RRI30  E370   RRI40  E38B   RRI50  E399   RRI60  E3A2
RRI70  E3A8   RRI80  E3BB   RRI90  E3C1   RRI99  E3C2
RRISN  E3C3
```

```
002                             ORG   :E401
003                     *
004                     *
005                     *
006                     ************************
007                     * RUN basiccmd RESTORE *
008                     ************************
009                     *
010 E401 2A9F02         RREST   LHLD  :029F     Get startaddr. textbuf
011 E404 222401                 SHLD  :0124     Store it in DATAP
012 E407 AF                     XRA   A
013 E408 3D                     DCR   A
014 E409 C352E4                 JMP   :E452     Set DATAQ=FF
015                     *
016                     ******************
017                     * RUN RANDOMISE *
018                     ******************
019                     *
020                     * Returns a hardware random number 0 < R < 1.
021                     *
022                     * Exit: BC preserved. AFDEHL corrupted.
023                     *       Result in MACC.
024                     *
025 E40C C5             RRAND   PUSH  B
026 E40D 2100FD                 LXI   H,:FD00   Addr PORI
027 E410 CD5FD6                 CALL  :D65F     BC=0, E=0, #40 or #80
028 E413 1601                   MVI   D,:01
029                     *
030 E415 3A9302         RMI10   LDA   :0293     Get RNDLY (0 by C72E)
031 E418 CD58D6                 CALL  :D658     Delay, A=M XOR E
032 E41B 07                     RLC
033 E41C 07                     RLC
034
035                     * Now CY is 0 if E=#40 and CY = bit 6 of FD00
036                     * (hardware random) if E=0 or E=#80.
037
038 E41D 7A             RMI20   MOV   A,D
039 E41E 17                     RAL             RAL D
040 E41F 57                     MOV   D,A
041 E420 79                     MOV   A,C
042 E421 17                     RAL             RAL C
043 E422 4F                     MOV   C,A
044 E423 78                     MOV   A,B
045 E424 17                     RAL             RAL B
046 E425 47                     MOV   B,A
047 E426 B7                     ORA   A
048 E427 F215E4                 JP    :E415     Again if B<#80 (must be
049                                               normalised FPT nr)
050
051                     * Result in MACC:
052
053 E42A 3E01                   MVI   A,:01     Set exp.byte for 1 < R < 2
054 E42C E7                     RST   4         Copy A,B,C,D to MACC
055 E42D 12                     DATA  :12
056 E42E 21F1D0                 LXI   H,:D0F1   Addr FPT(-1)
057 E431 E7                     RST   4         Add -1 to MACC
058 E432 00                     DATA  :00       Now: 0 < R < 1
059 E433 C1                     POP   B
060 E434 B7                     ORA   A         No special action
061 E435 C9                     RET
062                     *
063                     *
```

```
064                     ******************
065                     * cont. of 0E3DC *
066                     ******************
067                     *
068 E436 23             MPT36   INX   H
069 E437 7E                     MOV   A,M       Get length dataline
070 E438 323401                 STA   :0134     Store it in EFECT
071 E43B 23                     INX   H         Pnts to start data bytes
072 E43C 223201                 SHLD  :0132     Startaddr string in EFEPT
073 E43F 00                     NOP
074 E440 00                     NOP
075 E441 E1                     POP   H
076 E442 D1                     POP   D
077 E443 1E00                   MVI   E,:00
078 E445 F1                     POP   PSW
079 E446 C9                     RET
080                     *
081                     ****************************
082                     * PREPARE GETTING INPUTS *
083                     ****************************
084                     *
085 E447 221D01         MPT5A   SHLD  :011D     Store SP+2 in ERSSP
086 E44A 3EFF                   MVI   A,:FF
087 E44C 321701                 STA   :0117     Set flag for running inputs
088 E44F C3D0E3                 JMP   :E3D0     Print '?', input a textline
089                     *
090                     ******************
091                     * cont. of 0E401 *
092                     ******************
093                     *
094 E452 322301         MPT32   STA   :0123     Save offset next char to
095 E455 C9                     RET             encode
096                     *
097 E456 FF                     DATA  :FF
098 E457 FF                     DATA  :FF
099 E458 FF                     DATA  :FF
100 E459 FF                     DATA  :FF
101                     *
102                     ********************
103                     * RUN basiccmd LET *
104                     ********************
105                     *
106                     * Valid as direct command or in program. Computes
107                     * variable pointer or variable reference and
108                     * checks type. 'LET' can be explicitly or
109                     * implicitly given.
110                     *
111                     * Entry: BC: Points to program line.
112                     * Exit:  BC: Updated.
113                     *        A:  Type of variable.
114                     *        FDEHL corrupted.
115                     *
116                     RLETX
117 E45A CD63E9         RLETI   CALL  :E963     Get varptr in HL, T/L in A
118 E45D E630                   ANI   :30       Type only
119 E45F F5                     PUSH  PSW       Save type
120 E460 FE20                   CPI   :20       String type?
121 E462 C294E4                 JNZ   :E494     Jump if INT/FPT
122
123                     * If string type:
124
125 E465 E5             RLT10   PUSH  H         Save varptr
```

```
126 E466 7E                 MOV   A,M       )
127 E467 23                 INX   H         )
128 E468 66                 MOV   H,M       ) Addr string in HL
129 E469 6F                 MOV   L,A
130 E46A E3                 XTHL            ) Save stringaddr
131 E46B E5                 PUSH  H         ) and varptr
132 E46C CD9DE7             CALL  :E79D     Get value being assigned
133 E46F 7B                 MOV   A,E       Get status
134 E470 FE02               CPI   :02
135 E472 CA7FE4             JZ    :E47F     Jump if temp on heap
136 E475 7E                 MOV   A,M       Get string length
137 E476 EB                 XCHG
138 E477 CD8BD1             CALL  :D18B     Get place in heap for string
139 E47A E5                 PUSH  H
140 E47B CD72D1             CALL  :D172     Transfer string into heap
141 E47E E1                 POP   H
142 E47F EB         RLT20   XCHG            Pntr to string in DE
143 E480 E1                 POP   H         Pntr to variable in HL
144 E481 73                 MOV   M,E       ) Stringpntr in variable
145 E482 23                 INX   H         )
146 E483 72                 MOV   M,D       )
147
148                 * Entry from scratch:
149
150 E484 2A9F02     RLT25   LHLD  :029F     Get startaddr. textbuf
151 E487 EB                 XCHG            in DE
152 E488 E1                 POP   H         Get pntr old value
153 E489 7C                 MOV   A,H
154 E48A B5                 ORA   L
155 E48B C414DE             CNZ   :DE14     If <>0: Test if on heap
156 E48E DC87D1             CC    :D187     Then clear string in heap
157 E491 C3B2E4             JMP   :E4B2     Ready
158
159                 * If FPT/INT type:
160
161 E494 E5           RLT30   PUSH  H
162 E495 CD19E8             CALL  :E819     Eval numeric arguments
163 E498 7C                 MOV   A,H
164 E499 B5                 ORA   L
165 E49A C2A3E4             JNZ   :E4A3     Copy num expr if not in MACC
166 E49D E1                 POP   H
167 E49E E7                 RST   4         Copy MACC to variable
168 E49F 0F                 DATA  :0F
169 E4A0 C3B2E4             JMP   :E4B2     Ready
170
171                 * If just constant or variable:
172
173 E4A3 D1           RLT50   POP   D         Get pntr to variable
174 E4A4 D5                 PUSH  D
175 E4A5 C5                 PUSH  B
176 E4A6 0604               MVI   B,:04     Nr of bytes
177 E4A8 7E         RLT60   MOV   A,M       Get byte
178 E4A9 12                 STAX  D         Store it in variable
179 E4AA 23                 INX   H
180 E4AB 13                 INX   D
181 E4AC 05                 DCR   B         Decr byte count
182 E4AD C2A8E4             JNZ   :E4A8     Next byte if not ready
183 E4B0 C1                 POP   B
184 E4B1 E1                 POP   H
185
186                 * Ready:
187
```

```
188 E4B2 F1          RLT90   POP   PSW
189 E4B3 C9                  RET
190
191                  * Entry from READ/INPUT for FPT/INT:
192
193 E4B4 F5          RLT70   PUSH  PSW
194 E4B5 C394E4              JMP   :E494     Value to variable
195
196                  * Entry from READ/INPUT for STR:
197
198 E4B8 F5          RLT80   PUSH  PSW
199 E4B9 C365E4              JMP   :E465     Stringvalue to variable
200                  *
201                  **********************
202                  * RUN basiccmd SOUND *
203                  **********************
204                  *
205                  * Formats: SOUND <CHAN><ENV><VOL><TG><FREQ>.
206                  *          SOUND <CHAN> OFF.
207                  *          SOUND OFF.
208                  *
209                  * Entry: BC: Points to program line.
210                  * Exit:  BC updated, AFDEHL corrupted.
211                  *
212 E4BC 0A          RSOUND  LDAX  B         Get 1st expr.
213 E4BD FEFF                CPI   :FF       Is it sound OFF?
214 E4BF CA01E5              JZ    :E501     Then turn all sound off
215 E4C2 3E02                MVI   A,:02
216 E4C4 CD43E7              CALL  :E743     Get channelnr (0,1,2)
217 E4C7 21C201              LXI   H,:01C2   Startaddr SCB0
218 E4CA 110E00              LXI   D,:000E   Length SCB
219 E4CD F5                  PUSH  PSW       Save channelnr
220 E4CE 3D          SND10   DCR   A
221 E4CF FAD6E4              JM    :E4D6     If chan.0 or ready
222 E4D2 19                  DAD   D         Absolute addr SCB in HL
223 E4D3 C3CEE4              JMP   :E4CE     If chan.2
224 E4D6 D1          SND20   POP   D         Channelnr in D
225 E4D7 CD1FE5              CALL  :E51F     Set up SCB <ENV><VOL>
226 E4DA DAFCE4              JC    :E4FC     If channel to be OFF
227 E4DD 3E03                MVI   A,:03
228 E4DF CD43E7              CALL  :E743     Get <TG>
229 E4E2 F5                  PUSH  PSW
230 E4E3 E601                ANI   :01       Tremoloflag only
231 E4E5 77                  MOV   M,A       Into SCB
232 E4E6 23                  INX   H
233 E4E7 00                  NOP
234 E4E8 00                  NOP
235 E4E9 F1                  POP   PSW       Restore <TG>
236 E4EA E602                ANI   :02       Glissandoflag only
237 E4EC 1F                  RAR
238 E4ED 3C                  INR   A
239 E4EE 23                  INX   H         1 free byte
240 E4EF 77                  MOV   M,A       Glissandoflag in SCB
241 E4F0 23                  INX   H         )
242 E4F1 23                  INX   H         ) Ignore current period
243 E4F2 23                  INX   H         )
244 E4F3 E5                  PUSH  H         Save pntr to reqd period
245 E4F4 CDF8E6              CALL  :E6F8     Get <period>
246 E4F7 EB                  XCHG            in DE
247 E4F8 E1                  POP   H
248 E4F9 73                  MOV   M,E       ) Reqd period in SCB
249 E4FA 23                  INX   H         )
```

```
250 E4FB 72                 MOV   M,D        )
251 E4FC CD96D9       SND70 CALL  :D996      Enable sound interrupts
252 E4FF B7                 ORA   A          No special action
253 E500 C9                 RET
254
255                   * If SOUND OFF:
256
257 E501 03           SND80 INX   B
258 E502 CD8FD9             CALL  :D98F      Disable sound interrupts
259 E505 C5                 PUSH  B
260 E506 CDA6D8             CALL  :D8A6      Stop oscillators
261 E509 C1                 POP   B
262 E50A B7                 ORA   A          No special action
263 E50B C9                 RET
264                   *
265                   **********************
266                   * RUN basiccmd NOISE *
267                   **********************
268                   *
269                   * Sets up a noise control block.
270                   * Formats: NOISE <ENV><VOL>.
271                   *          NOISE OFF.
272                   *
273                   * Entry: BC: Points to expression.
274                   * Exit:  Noise off: BC updated, DE preserved,
275                   *                   AFHL corrupted.
276                   *        Noise on:  BC updated, AFDEHL corrupted.
277                   *
278 E50C 21EC01        RNOISE LXI   H,:01EC   Startaddr NCB
279 E50F 1603                 MVI   D,:03     Channelnr.3
280 E511 CD1FE5               CALL  :E51F     Set up NCB
281 E514 DAFCE4               JC    :E4FC     If channel to be off
282 E517 3600                 MVI   M,:00     No tremolo
283 E519 23                   INX   H
284 E51A 3600                 MVI   M,:00     Current volume = 0
285 E51C C3FCE4               JMP   :E4FC     Enable sound interrupts
286                   *
287                   ****************
288                   * SET ENVELOPE *
289                   ****************
290                   *
291                   * Sets up a sound or noise control block for
292                   * a channel.
293                   *
294                   * Entry: D:  Channelnumber (0,1,2 or 3).
295                   *        BC: Points to programline.
296                   *        HL: Points to startaddr SCB/NCB.
297                   * Exit:  If sound/noise off (CY=1):
298                   *        FF in 1st byte of SCB/NCB.
299                   *            HL: points to 1st byte SCB/NCB.
300                   *            BC: points to 2nd byte SCB/NCB.
301                   *            DE preserved, AF corrupted.
302                   *        If sound/noise on (CY=0):
303                   *        00 in 1st byte SCB/NCB.
304                   *            HL: Points after free byte.
305                   *            DE: Envelopeaddr +1.
306                   *            BC: Points beyond volume.
307                   *            AF: corrupted.
308                   *
309 E51F CD8FD9        SNGEV  CALL  :D98F      Disable sound interrupts
310 E522 3600                 MVI   M,:00     00 in 1st byte SCB/NCB
311 E524 0A                   LDAX  B          Get 1st byte from progr.line
```

```
312 E525 FEFF              CPI   :FF
313 E527 CA52E5            JZ    :E552     Jump if channel to be off
314
315                * If channel on:
316
317 E52A 23                INX   H         Pntr to next byte of block
318 E52B E5                PUSH  H         Save pntr
319 E52C 3E01              MVI   A,:01
320 E52E CD43E7            CALL  :E743     Get envelopenr in A (0,1)
321 E531 21F501            LXI   H,:01F5   Addr envelope area
322 E534 0F                RRC
323 E535 0F                RRC
324 E536 5F                MOV   E,A       DE=64*A
325 E537 1600              MVI   D,:00
326 E539 19                DAD   D         Add offset for env area
327 E53A EB                XCHG            Pntr to envelope in DE
328 E53B E1                POP   H         Get input pntr SCB/NCB
329 E53C 73                MOV   M,E       ) Envelope addr in block
330 E53D 23                INX   H         )
331 E53E 72                MOV   M,D       )
332 E53F 23                INX   H
333 E540 13                INX   D
334 E541 73                MOV   M,E       ) Env addr +1 in block
335 E542 23                INX   H         )
336 E543 72                MOV   M,D       )
337 E544 23                INX   H
338 E545 3E0F              MVI   A,:0F
339 E547 CD43E7            CALL  :E743     Get volume multiplier in
340                                        A (0-15)
341 E54A 07                RLC             )
342 E54B 07                RLC             ) *8
343 E54C 07                RLC             )
344 E54D 77                MOV   M,A       Vol.*8 in block
345 E54E 23                INX   H         Pntr to basic volume
346 E54F 23                INX   H         Pntr to tremoloflag
347 E550 B7                ORA   A         Return 'channel on'
348 E551 C9                RET
349
350                * If channel to be off:
351
352 E552 03       SNG10    INX   B
353 E553 35                DCR   M         FF in 1st byte SCB/NCB
354 E554 7A                MOV   A,D       Get channelnr
355 E555 FE03              CPI   :03
356 E557 CA63E5            JZ    :E563     Jump if noisechannel
357 E55A 0F                RRC             ) Find disable data for
358 E55B 0F                RRC             ) chan. 0-2
359 E55C F636              ORI   :36       )
360 E55E 3206FC            STA   :FC06     Load sound cmd word
361 E561 37                STC             Return 'channel off'
362 E562 C9                RET
363
364                * If noise to be off:
365
366 E563 3A9502   SNG20    LDA   :0295     Get volume osc.2 + noise
367 E566 E60F              ANI   :0F       Vol. noise = 0
368 E568 329502            STA   :0295     Update POR1M
369 E56B 3205FD            STA   :FD05     and POR1
370 E56E 37                STC             Return 'channel off'
371 E56F C9                RET
372                *
373                *
```

```
                        **************************
                        * RUN basiccmd ENVELOPE  *
                        **************************
                        *
                        * Load envelope table 0 or 1 with data.
                        * Formats: ENVELOPE <ENV> (<V>,<T>;) <V>,<T> FF.
                        *          ENVELOPE <ENV> (<V>,<T>;) <V>.
                        *
E570 3E01               RENV    MVI   A,:01
E572 CD43E7                     CALL  :E743     Get envelope number (0,1)
E575 0F                         RRC
E576 0F                         RRC
E577 5F                         MOV   E,A       ) Offset for env addr
E578 1600                       MVI   D,:00     )
E57A 21F501                     LXI   H,:01F5   Addr env storage area
E57D 19                         DAD   D         Startaddr table in HL
E57E 3600                       MVI   M,:00     00 in 1st field
E580 23                         INX   H
E581 0A                         LDAX  B         Get length of expr
E582 03                         INX   B
E583 3C                         INR   A
E584 57                         MOV   D,A       Nr of complete entries in D
E585 15                 REN10   DCR   D
E586 CA9DE5                     JZ    :E59D     If all entries done
E589 3E10                       MVI   A,:10
E58B CD43E7                     CALL  :E743     Get volume (0-16)
E58E 77                         MOV   M,A       Into env table
E58F 23                         INX   H
E590 CD1DE7                     CALL  :E71D     Get time
E593 D601                       SUI   :01       Min. value is 1
E595 DA15DA                     JC    :DA15     Run error 'NUMBER OUT OF
                                                RANGE' if time=0
E598 77                         MOV   M,A       Time into env table
E599 23                         INX   H
E59A C385E5                     JMP   :E585     Next <V>,<T>
E59D 36FF               REN15   MVI   M,:FF     FF as last in env table
E59F 0A                         LDAX  B         Get last expr
E5A0 03                         INX   B
E5A1 FEFF                       CPI   :FF       End marker?
E5A3 CAB0E4                     JZ    :E4B0     Then quit
E5A6 0B                         DCX   B
E5A7 3E10                       MVI   A,:10
E5A9 CD43E7                     CALL  :E743     Get final volume <V>
E5AC 77                         MOV   M,A       <V> into env table
E5AD 23                         INX   H
E5AE 36FF                       MVI   M,:FF     Time = forever
E5B0 B7                 REN20   ORA   A         No special action
E5B1 C9                         RET
                        *
                        **************************
                        * RUN basiccmd CURSOR    *
                        **************************
                        *
E5B2 CDF3E5             RCURS   CALL  :E5F3     Evaluate coordinate
E5B5 67                         MOV   H,A       Y-coord in H
E5B6 EF                         RST   5         Set cursor position
E5B7 09                         DATA  :09
E5B8 C3C9E5                     JMP   :E5C9     Evt run screen error
                        *
                        **********************
                        * RUN basiccmd MODE  *
                        **********************
```

```
436                     *
437                     * Entry: BC: Points to program line.
438                     *
439 E5BB 0A             RMODE   LDAX  B         Get reqd mode in A
440 E5BC 03                     INX   B
441 E5BD C3B5CE                 JMP   :CEB5     Change screen mode
442                     *
443 E5C0 C9             LOE99   RET
444                     *
445                     **********************
446                     * RUN basiccmd DOT *
447                     **********************
448                     *
449 E5C1 CDF9E5         RDOT    CALL  :E5F9     Eval dot addr + colour
450 E5C4 C5                     PUSH  B
451 E5C5 4B                     MOV   C,E       Colour in C
452 E5C6 EF                     RST   5         Draw dot on screen
453 E5C7 1E                     DATA  :1E
454 E5C8 C1             RDT10   POP   B
455 E5C9 DA02E6         RDT20   JC    :E602     Jump if screen error
456 E5CC B7                     ORA   A         No special action
457 E5CD C9                     RET
458                     *
459                     ************************
460                     * RUN basiccmd DRAW *
461                     ************************
462                     *
463 E5CE CDE0E5         RDRAW   CALL  :E5E0     Eval begin/end addr +
464                                             colour
465 E5D1 E3                     XTHL
466 E5D2 EF                     RST   5         Draw a line on screen
467 E5D3 21                     DATA  :21
468 E5D4 C3C8E5                 JMP   :E5C8     Evt. run screen error
469                     *
470                     ************************
471                     * RUN basiccmd FILL *
472                     ************************
473                     *
474 E5D7 CDE0E5         RFILL   CALL  :E5E0     Eval coordinates, colour
475 E5DA E3                     XTHL
476 E5DB EF                     RST   5         Fill a rectangular area
477 E5DC 24                     DATA  :24
478 E5DD C3C8E5                 JMP   :E5C8     Evt run screen error
479                     *
480                     ************************************
481                     * EVALUATE 2 COORDINATES + COLOUR *
482                     ************************************
483                     *
484 E5E0 CDF3E5         R2COC   CALL  :E5F3     Get 1st coordinate
485 E5E3 E3                     XTHL            X-coord on stack
486 E5E4 E5                     PUSH  H
487 E5E5 5F                     MOV   E,A       Y-coord in E
488 E5E6 CDF3E5                 CALL  :E5F3     Get 2nd coordinate,
489                                             x-coord in HL
490 E5E9 57                     MOV   D,A       Y-coord in D
491 E5EA CDFDE5                 CALL  :E5FD     Get colour in A
492 E5ED C5                     PUSH  B
493 E5EE D5                     PUSH  D
494 E5EF EB                     XCHG
495 E5F0 C1                     POP   B
496 E5F1 E1                     POP   H
497 E5F2 C9                     RET
```

```
498                     *
499                     ************************
500                     * EVALUATE COORDINATE *
501                     ************************
502                     *
503 E5F3 CDF8E6         RCOOR   CALL   :E6F8      Get x-coord in HL
504 E5F6 C31DE7                 JMP    :E71D      Get y-coord in A
505                     *
506                     ***********************************
507                     * EVALUATE COORDINATE + COLOUR *
508                     ***********************************
509                     *
510 E5F9 CDF3E5         RCOCO   CALL   :E5F3      Get x-coord in HL
511 E5FC 5F                     MOV    E,A        Y-coord in E
512 E5FD 3E17           RCOL    MVI    A,:17
513 E5FF C343E7                 JMP    :E743      Get colour (0-23) in A
514                     *
515                     *
516                     *
517 E602                        END
```

```
*****************************
* S Y M B O L   T A B L E *
*****************************
```

L0E99	E5C0	MPT32	E452	MPT36	E436	MPT5A	E447
R2COC	E5E0	RCOCO	E5F9	RCOL	E5FD	RCOOR	E5F3
RCURS	E5B2	RDOT	E5C1	RDRAW	E5CE	RDT10	E5C8
RDT20	E5C9	REN10	E585	REN15	E59D	REN20	E5B0
RENV	E570	RFILL	E5D7	RLETI	E45A	RLETX	E45A
RLT10	E465	RLT20	E47F	RLT25	E484	RLT30	E494
RLT50	E4A3	RLT60	E4A8	RLT70	E4B4	RLT80	E4B8
RLT90	E4B2	RMI10	E415	RMI20	E41D	RMODE	E5BB
RNOISE	E50C	RRAND	E40C	RREST	E401	RSOUND	E4BC
SND10	E4CE	SND20	E4D6	SND70	E4FC	SND80	E501
SNG10	E552	SNG20	E563	SNGEV	E51F		

```
002                             ORG   :E602
003                     *
004                     *
005                     *
006                     **********************
007                     * RUN SCREEN ERROR *
008                     **********************
009                     *
010                     * Entry: A: Error code: 01: Off screen.
011                     *                       02: Colour not available.
012 E602 FE01           SCRER   CPI   :01       Code is 1?
013 E604 3E11                   MVI   A,:11
014 E606 CAF5D9                 JZ    :D9F5      Then run error 'OFF SCREEN'
015 E609 3E10                   MVI   A,:10      Else: run error
016 E60B C3F5D9                 JMP   :D9F5      'COLOUR NOT AVAILABLE'
017                     *
018                     ************************
019                     * RUN basiccmd COLORT *
020                     ************************
021                     *
022 E60E CD1CE6         RCOLT   CALL  :E61C      Get colours in scratch area
023 E611 EF                     RST   5          Set text colours
024 E612 06                     DATA  :06
025 E613 B7                     ORA   A          No special action
026 E614 C9                     RET
027                     *
028                     ************************
029                     * RUN basiccmd COLORG *
030                     ************************
031                     *
032 E615 CD1CE6         RCOLG   CALL  :E61C      Get colours in scratch area
033 E618 EF                     RST   5          Set graphic colours
034 E619 1B                     DATA  :1B
035 E61A B7                     ORA   A          No special action
036 E61B C9                     RET
037                     *
038                     ************************************
039                     * GET 4 COLOURS INTO SCRATCH AREA *
040                     ************************************
041                     *
042                     * Colour data from a program line are stored in
043                     * scratch area SCOLT/SCOLG (#0119-011C).
044                     *
045                     * Exit: HL: Points to start scratch area.
046                     *
047 E61C 211901         R4COL   LXI   H,:0119    Startaddr SCOLT/SCOLG area
048 E61F E5                     PUSH  H
049 E620 3E0F           R4C10   MVI   A,:0F
050 E622 CD43E7                 CALL  :E743      Get one colour (0-15)
051 E625 77                     MOV   M,A        Store it in scratch area
052 E626 23                     INX   H
053 E627 7D                     MOV   A,L
054 E628 FE1D                   CPI   :1D        4 colours done?
055 E62A C220E6                 JNZ   :E620      Next if not
056 E62D E1                     POP   H
057 E62E C9                     RET
058                     *
059                     **********************
060                     * RUN basiccmd DIM *
061                     **********************

                          Entry: BC: points to program line.
```

```
064                     *
065 E62F 0A             RDIM    LDAX  B          Get nr of items
066 E630 03                     INX   B
067 E631 B7             RDM05   ORA   A
068 E632 C8                     RZ                Abort if no items or ready
069 E633 3D                     DCR   A           Item count
070 E634 F5                     PUSH  PSW         Preserve count
071 E635 CD5AE9                 CALL  :E95A       Get pnr to array in HL;
072                                               type in A
073 E638 E5                     PUSH  H           Preserve pntr
074 E639 CD5CCE                 CALL  :CE5C       Erase array if existing
075 E63C E630                   ANI   :30         Get type only
076 E63E 110400                 LXI   D,:0004     Length array element if
077                                               FPT/INT
078 E641 FE20                   CPI   :20         String type?
079 E643 C248E6                 JNZ   :E648       Jump if not
080 E646 1E02                   MVI   E,:02       Length STR array element
081 E648 0A             RDM10   LDAX  B           Get number of elements
082 E649 03                     INX   B
083 E64A 67                     MOV   H,A         ) In H and in L
084 E64B 6F                     MOV   L,A         )
085 E64C EB                     XCHG
086
087                     * Calculate total required length:
088
089 E64D CD4FE7         RDM20   CALL  :E74F       Get length next dimension
090 E650 F5                     PUSH  PSW         Remember it
091 E651 3C                     INR   A           Length +1
092 E652 CDF0ED                 CALL  :EDF0       Calc reqd space
093 E655 DA15DA                 JC    :DA15       Run error 'NUMBER OUT
094                                               OF RANGE' if total space
095                                               > 64K
096 E658 15                     DCR   D           nr of elements -1
097 E659 C24DE6                 JNZ   :E64D       Next element if not ready
098
099                     * Find space in heap:
100
101 E65C 25                     DCR   H
102 E65D 24                     INR   H
103 E65E FA15DA                 JM    :DA15       Run error 'NUMBER OUT OF
104                                               RANGE' if > 32K reserved
105 E661 19                     DAD   D
106 E662 23                     INX   H           Size of space reqd in HL
107 E663 D5                     PUSH  D
108 E664 EB                     XCHG
109 E665 CD8BE6                 CALL  :E68B       Get space of size needed
110 E668 D1                     POP   D
111 E669 73                     MOV   M,E         Store nr of elements
112 E66A 19                     DAD   D           Last element
113
114                     * Elements into heap:
115
116 E66B F1             RDM30   POP   PSW         Get length 1 element
117 E66C 77                     MOV   M,A         Store it in memory
118 E66D 2B                     DCX   H
119 E66E 1D                     DCR   E
120 E66F C26BE6                 JNZ   :E66B       Next element to memory
121                     *
122 E672 EB                     XCHG
123 E673 E1                     POP   H           Get pntr to array
124 E674 73                     MOV   M,E         )
125 E675 23                     INX   H           ) Set pointer
```

```
126 E676 72                     MOV   M,D       )
127 E677 F1                     POP   PSW       Get item count in A
128 E678 C331E6                 JMP   :E631     Next item
129                     *
130                     ****************************
131                     * part of RUN TALK (0EE94) *
132                     ****************************
133                     *
134                     * Entry: A: Code for osc.channel SHR 1.
135                     *
136 E67B 29             MPT47   DAD   H
137 E67C 119402         RTK50   LXI   D,:0294   Addr volumes osc. 0,1
138 E67F E601                   ANI   :01       Code SHR 1 only
139 E681 83                     ADD   E
140 E682 5F                     MOV   E,A       DE=#0294 for osc.0,1;
141                                             DE=#0295 for osc.2,N
142 E683 7C                     MOV   A,H       Get mask
143 E684 2F                     CMA             Complement it
144 E685 EB                     XCHG            Mask + vol in DE, addr
145                                             POROM/POR1M in HL
146 E686 A6                     ANA   M         Part to be preserved from
147                                             old POROM/POR1M
148 E687 B3                     ORA   E         Add new volume
149 E688 C340EA                 JMP   :EA40     Continu
150                     *
151                     ************************
152                     * REQUEST HEAP SPACE *
153                     ************************
154                     *
155                     * Part of Run 'DIM' (0E665).
156                     * Requests space from Heap and fills it with
157                     * zeroes.
158                     *
159                     * Entry: DE: Size needed.
160                     * Exit:  HL: Points to data area (after length
161                     *            bytes).
162                     *        AFDE corrupted.
163                     *
164 E68B 15             ZHREQ   DCR   D
165 E68C 14                     INR   D
166 E68D FA15DA                 JM    :DA15     Run error 'NUMBER OUT OF
167                                             RANGE' if >32K reqd
168 E690 CDC5D1                 CALL  :D1C5     Run Heap request
169 E693 23                     INX   H
170 E694 23                     INX   H         HL pnts after length byte
171 E695 E5                     PUSH  H
172 E696 EB                     XCHG            Start data area in DE
173 E697 19                     DAD   D         End area in HL
174 E698 AF                     XRA   A
175 E699 CD7CDE                 CALL  :DE7C     Load bank with '0'
176 E69C E1                     POP   H
177 E69D C9                     RET
178                     *
179                     ********************
180                     * RUN basiccmd UT *
181                     ********************
182                     *
183                     * Valid as direct command only.
184                     *
185 E69E AF             RUT     XRA   A
186 E69F 32B902                 STA   :02B9     Enable complete keyb scan
187 E6A2 CF                     RST   1         Go to utility
```

```
188 E6A3 09                 DATA  :09
189                 *
190                 ************************
191                 * RUN basiccmd CALLM *
192                 ************************
193                 *
194 E6A4 21B3E6      RCALM   LXI   H,:E6B3   Returnaddr from Utility
195 E6A7 E5                 PUSH  H         on stack
196 E6A8 CDF8E6             CALL  :E6F8     Get UT addr in HL
197 E6AB E5                 PUSH  H         UT addr on stack
198 E6AC 0A                 LDAX  B         Get next expr
199 E6AD FEFF               CPI   :FF       End marker?
200 E6AF C263E9             JNZ   :E963     If not: Get varptr of given
201                                         variable in HL, T/L in A
202 E6B2 03                 INX   B
203 E6B3 B7         RCM10   ORA   A         No special action
204 E6B4 C9                 RET             On entry: Goto UT addr
205                                         On exit: Back to Basic
206                                         monitor
207                 *
208                 ************************
209                 * RUN basiccmd CLEAR *
210                 ************************
211                 *
212 E6B5 CD7FD8      RCLEAR  CALL  :D87F     Get space reqd in HL
213                                         (CY=1 if > 32K)
214 E6B8 CDBBCE             CALL  :CEBB     Must be >=4 bytes, else run
215                                         error 'NUMBER OUT OF RANGE'
216 E6BB EB                 XCHG
217 E6BC 229D02             SHLD  :029D     Set Heap size
218 E6BF 2A9F02             LHLD  :029F     Get startaddr. textbuf
219 E6C2 E5                 PUSH  H
220 E6C3 CD23CB             CALL  :CB23     Empty Heap + symtab
221 E6C6 D5                 PUSH  D
222 E6C7 CD95D1             CALL  :D195     Set Heap to all available
223 E6CA E1                 POP   H
224 E6CB C314D2             JMP   :D214     Continu
225                 *
226                 ********************************
227                 * Run basiccmds TRON - TROFF *
228                 ********************************
229                 *
230                 * Sets or resets the trace flag.
231                 *
232                 * RTRON: Set trace flag.
233                 * RTROF: Reset trace flag.
234                 *
235                 * Entry: none.
236                 * Exit:  Z=1: Flag reset.
237                 *        Z=0: Flag set.
238                 *
239 E6CE 3EFF       RTRON   MVI   A,:FF
240 E6D0 321501     RTR10   STA   :0115     Set trace flag
241 E6D3 B7                 ORA   A         No special action
242 E6D4 C9                 RET
243                 *
244 E6D5 AF         RTROF   XRA   A
245 E6D6 C3D0E6             JMP   :E6D0     Reset trace flag
246                 *
247                 ********************
248                 * READ LINENUMBER *
249                 ********************
```

```
                        *
                        * Entry: BC: Points to linenumber.
                        * Exit:  Z=0: Linenumber in HL.
                        *        Z=1: HL preserved.
                        *        BC updated, DE preserved, AF corrupted.
                        *
E6D9 E5         RLN     PUSH  H
E6DA 0A                 LDAX  B
E6DB 03                 INX   B
E6DC 67                 MOV   H,A       )
E6DD 0A                 LDAX  B         ) Get linenr in HL
E6DE 03                 INX   B         )
E6DF 6F                 MOV   L,A       )
E6E0 B4                 ORA   H
E6E1 CAE5E6             JZ    :E6E5     Abort if linenr is 0
E6E4 E3                 XTHL            Linenr on stack
E6E5 E1         RLN10   POP   H         Old HL or linenr in HL
E6E6 C9                 RET
                        *
                        ***********************************************
                        * READ LINENUMBER AND FIND IT IN TEXTBUFFER *
                        ***********************************************
                        *
                        * Entry: BC: Points to linenumber.
                        * Exit:  BC updated, DE preserved, AF corrupted
                        *        (RLNF) or preserved (RLNFI).
                        *        HL: Points to 1st linenr >= reqd. number.
                        *        CY=1: Linenumber found.
                        *        CY=0: Not found.
                        *
E6E7 CDD9E6     RLNF    CALL  :E6D9     Get linenr in HL
E6EA C3F6CA             JMP   :CAF6     Find it in textbuffer

                        * Idem as RLNF, but with error reporting:

E6ED F5         RLNFI   PUSH  PSW
E6EE CDE7E6             CALL  :E6E7     Read linenr and find it
E6F1 3E04               MVI   A,:04
E6F3 D2F5D9             JNC   :D9F5     Run error 'UNDEFINED NUMBER'
                                        if not found
E6F6 F1                 POP   PSW
E6F7 C9                 RET
                        *
                        ***********************************************
                        * RUN A INT EXPRESSION WITH 2-BYTE RESULT *
                        ***********************************************
                        *
                        * Evaluates a 16-bit INT expression (in range 0-
                        * FFFF). The result is in HL.
                        *
                        * Entry: BC: Points to expression.
                        * Exit:  HL: Result.
                        *        BC updated, AFDE corrupted.
                        *
E6F8 F5         REXI2   PUSH  PSW
E6F9 D5                 PUSH  D
E6FA CD19E8             CALL  :E819     Eval arguments in num expr
                                        Result in MACC or in WORKE
E6FD 7C                 MOV   A,H
E6FE B5                 ORA   L
E6FF CA10E7             JZ    :E710     Jump if result in MACC
```

```
312                     * If result in WORKE:
313
314 E702 7E                     MOV   A,M       )
315 E703 23                     INX   H         ) Check if > 2 bytes
316 E704 B6                     ORA   M         )
317 E705 C215DA                 JNZ   :DA15     Then run error 'NUMBER OUT
318                                             OF RANGE'
319 E708 23                     INX   H
320 E709 7E                     MOV   A,M       )
321 E70A 23                     INX   H         ) Get result in HL
322 E70B 6E                     MOV   L,M       )
323 E70C 67                     MOV   H,A       )
324 E70D D1                     POP   D
325 E70E F1                     POP   PSW
326 E70F C9                     RET
327
328                     * If result in MACC:
329
330 E710 C5             RX210   PUSH  B
331 E711 E7                     RST   4         Copy MACC to reg A,B,C,D
332 E712 15                     DATA  :15
333 E713 B0                     ORA   B         Check if > 2 bytes
334 E714 C215DA                 JNZ   :DA15     Then run error 'NUMBER OUT
335                                             OF RANGE'
336 E717 6A                     MOV   L,D       ) Result in HL
337 E718 61                     MOV   H,C       )
338 E719 C1                     POP   B
339 E71A D1                     POP   D
340 E71B F1                     POP   PSW
341 E71C C9                     RET
342                     *
343                     ********************************
344                     * RUN A 1-BYTE INT EXPRESSION *
345                     ********************************
346                     *
347                     * Evaluates a 8-bit INT expression (range 0-
348                     * FF). Result in A.
349                     *
350                     * Entry: BC: Points to expression.
351                     * Exit:   A:  Result.
352                     *         BC updated, DEHL preserved.
353                     *
354 E71D D5             REXI1   PUSH  D
355 E71E E5                     PUSH  H
356 E71F CD19E8                 CALL  :E819     Eval arguments in num expr
357                                             Result in MACC or WORKE
358 E722 7C                     MOV   A,H
359 E723 B5                     ORA   L
360 E724 CA36E7                 JZ    :E736     If HL=0: Get result frm MACC
361
362                     * Result in WORKE:
363
364 E727 7E                     MOV   A,M       )
365 E728 23                     INX   H         )
366 E729 B6                     ORA   M         ) Check if > 1 byte
367 E72A 23                     INX   H         )
368 E72B B6                     ORA   M         )
369 E72C C215DA                 JNZ   :DA15     Then run error 'NUMBER OUT
370                                             OF RANGE'
371 E72F 23                     INX   H
372 E730 7E                     MOV   A,M       Get result in A
373 E731 E1                     POP   H
```

```
374 E732 D1                     POP   D
375 E733 C9                     RET
376
377                   * If result in MACC (also entry from REXF1):
378
379 E734 D5           RX110     PUSH  D
380 E735 E5                     PUSH  H
381 E736 E1           RX120     POP   H
382 E737 C5                     PUSH  B
383 E738 E7                     RST   4           Copy MACC to reg A,B,C,D
384 E739 15                     DATA  :15
385 E73A B0                     ORA   B           ) Check if > 1 byte
386 E73B B1                     ORA   C           )
387 E73C C215DA                 JNZ   :DA15       Then run error 'NUMBER OUT
388                                               OF RANGE'
389 E73F 7A                     MOV   A,D         Get result in A
390 E740 C1                     POP   B
391 E741 D1                     POP   D
392 E742 C9                     RET
393                   *
394                   ************************************************
395                   * RUN 1-BYTE INT EXPRESSION WITH LIMITED RANGE *
396                   ************************************************
397                   *
398                   * Entry: BC: Points to expression.
399                   *         A:  Range of arguments (<=FE).
400                   * Exit:  BC updated, DEHL preserved, F corrupted.
401                   *         A:  Result.
402                   *
403 E743 D5           REXIL     PUSH  D
404 E744 57                     MOV   D,A         Argument range in D
405 E745 CD1DE7                 CALL  :E71D       Get value of argument in A
406 E748 14                     INR   D
407 E749 BA                     CMP   D           Out of range ? Then run
408 E74A D215DA                 JNC   :DA15       error 'NUMBER OUT OF RANGE'
409 E74D D1                     POP   D
410 E74E C9                     RET
411                   *
412                   ************************************************
413                   * CHECK VARIABLE TYPE AND GET ITS INT VALUE *
414                   ************************************************
415                   *
416                   * Entry: BC: Points to expression.
417                   * Exit:  Error: If string type.
418                   *         If OK: Value in A (FPT: converted to INT).
419                   *                BC updated, DEHL preserved.
420                   *
421 E74F 0A           REX1      LDAX  B           Get var. type byte
422 E750 03                     INX   B
423 E751 FE20                   CPI   :20         String type?
424 E753 CA1ADA                 JZ    :DA1A       Then run error 'TYPE
425                                               MISMATCH'
426 E756 FE10                   CPI   :10         INT type?
427 E758 CA1DE7                 JZ    :E71D       Then get value in A
428
429                   * If FPT:
430
431 E75B CD08E8       REXF1     CALL  :E808       Get value in MACC
432 E75E E7                     RST   4           Change it to INT
433 E75F 48                     DATA  :48
434 E760 C334E7                 JMP   :E734       Get value in A
435                   *
```

```
436                     *=====================================*
437                     * RUN EXPRESSIONS WITH OPERATOR PREFIX *
438                     *=====================================*
439                     *
440                     * #E763-E8ED evaluate logical, FPT, INT or STR
441                     * expressions in 'operator prefix' format.
442                     *
443                     * Register allocation during operation:
444                     *    INT/FPT: D=0:   MACC empty.
445                     *             E:     Operator.
446                     *             HL=0:  Result in MACC.
447                     *             HL<>0: HL points to result.
448                     *    STR:     HL:    Points to string.
449                     *             E:     Type of string (constant [0],
450                     *                    variable [1], temporary [2]).
451                     *
452                     ***********************************
453                     * EVALUATE A LOGICAL EXPRESSION *
454                     ***********************************
455                     *
456 E763 1600        REXPL   MVI   D,:00     MACC free
457 E765 0A                 LDAX  B          Get byte
458 E766 E660               ANI   :60
459 E768 FE40               CPI   :40        String ?
460 E76A CABDE7             JZ    :E7BD      Then jump
461 E76D 0A                 LDAX  B          Get byte
462 E76E E61F               ANI   :1F
463 E770 FE18               CPI   :18        Relational operator?
464 E772 DA50E8             JC    :E850      If not: eval expr which
465                                          begins with num operator
466 E775 03                 INX   B
467 E776 FE1A               CPI   :1A        Bracket?
468 E778 CA63E7             JZ    :E763      Then ignore it
469
470                     * Logical AND or OR:
471
472 E77B F5                 PUSH  PSW        Preserve type of operation
473 E77C CD63E7             CALL  :E763      Get 1st operand
474 E77F F5                 PUSH  PSW        Preserve it
475 E780 CD63E7             CALL  :E763      Get 2nd operand
476 E783 D1                 POP   D          1st operand in D
477 E784 F5                 PUSH  PSW        Preserve 2nd operand
478 E785 A2                 ANA   D          AND operation
479 E786 5F                 MOV   E,A        Result in E
480 E787 F1                 POP   PSW        2nd operand in A
481 E788 B2                 ORA   D          OR operation
482 E789 57                 MOV   D,A        Result in D
483 E78A F1                 POP   PSW        Type of operation in F
484 E78B 7A                 MOV   A,D        Result OR in A
485 E78C EA90E7             JPE   :E790      Quit if OR
486 E78F 7B                 MOV   A,E        Result AND in A
487 E790 C9             RXL10   RET
488                     *
489                     *********************************
490                     * EVALUATE STRING EXPRESSION *
491                     *********************************
492                     *
493                     * This routine returns temporary strings before
494                     * they are really free.
495                     * The heap is cleared if it is a temporary string.
496                     *
497                     * Entry: BC: Points to expression.
```

```
498                     * Exit:  BC updated, AFD corrupted.
499                     *        HL: Points to string.
500                     *        E:  Status.
501                     *
502 E791 CD9DE7         REXSR   CALL   :E79D      Evaluate string expr
503 E794 7B                    MOV    A,E        Get status
504 E795 FE02                  CPI    :02        Temporary ?
505 E797 E5                    PUSH   H
506 E798 CC87D1                CZ     :D187      Then clear heap entry
507 E79B E1                    POP    H
508 E79C C9                    RET
509                     *
510                     *
511                     *
512 E79D                       END
```

```
*****************************
* S Y M B O L   T A B L E *
*****************************
```

MPT47	E67B	R4C10	E620	R4COL	E61C	RCALM	E6A4
RCLEAR	E6B5	RCM10	E6B3	RCOLG	E615	RCOLT	E60E
RDIM	E62F	RDM05	E631	RDM10	E648	RDM20	E64D
RDM30	E66B	REX1	E74F	REXF1	E75B	REXI1	E71D
REXI2	E6F8	REXIL	E743	REXPL	E763	REXSR	E791
RLN	E6D9	RLN10	E6E5	RLNF	E6E7	RLNFI	E6ED
RTK50	E67C	RTR10	E6D0	RTROF	E6D5	RTRON	E6CE
RUT	E69E	RX110	E734	RX120	E736	RX210	E710
RXL10	E790	SCRER	E602	ZHREQ	E68B		

```
002                             ORG   :E79D
003                     *
004                     *
005                     *
006                     ********************************************
007                     * EVALUATE ARGUMENTS IN STRING EXPRESSION *
008                     ********************************************
009                     *
010                     * Only '+' or compare with logical result is
011                     * allowed. The right-hand side of a string expres-
012                     * sion is evaluated. If it is not status 02, it is
013                     * moved into the Heap. The stringpointer is saved
014                     * at the varptr location.
015                     * If the variable had already an old value on the
016                     * heap, it is cleared, see further exit conditions.
017                     *
018                     * Entry: (BC): 1.....  Expr. begins with operator.
019                     *              01....  Variable reference.
020                     *              001...  Function call.
021                     *              else    Constant.
022                     * Exit:  BC updated, DEHL corrupted.
023                     *        A: Type (#20).
024                     *
025 E79D 0A             REXPS   LDAX  B         get 1st byte
026 E79E 07                     RLC
027 E79F DABDE7                 JC    :E7BD     Jump if 1st byte is operator
028 E7A2 07                     RLC
029 E7A3 DAB3E7                 JC    :E7B3     Jump if string variable
030 E7A6 07                     RLC
031 E7A7 DAD9E9                 JC    :E9D9     Jump if string function
032
033                     * If string constant:
034
035 E7AA 1E00                   MVI   E,:00     Status: constant
036 E7AC 03                     INX   B
037 E7AD 0A                     LDAX  B         Get string length
038 E7AE 60                     MOV   H,B       ) Stringpntr in HL
039 E7AF 69                     MOV   L,C       )
040 E7B0 C390E1                 JMP   :E190     Abort with BC pnts after STR
041
042                     * If string variable:
043
044 E7B3 CD63E9         RXS10   CALL  :E963     Get varptr in HL, T/L in A
045 E7B6 5E                     MOV   E,M       )
046 E7B7 23                     INX   H         ) Stringaddr in DE
047 E7B8 56                     MOV   D,M       )
048 E7B9 EB                     XCHG            and in HL
049 E7BA 1E01                   MVI   E,:01     Status: variable
050 E7BC C9                     RET
051
052                     * If string operation:
053
054 E7BD 0A             ROSTR   LDAX  B         Get 1st byte
055 E7BE 03                     INX   B
056 E7BF F5                     PUSH  PSW
057 E7C0 D5                     PUSH  D
058 E7C1 CD9DE7                 CALL  :E79D     Evaluate string expression
059 E7C4 F1                     POP   PSW
060 E7C5 57                     MOV   D,A
061 E7C6 F1                     POP   PSW
062 E7C7 E5                     PUSH  H         Remember it
063 E7C8 53                     MOV   D,E       Type in D
```

```
064 E7C9 F5                PUSH  PSW
065 E7CA D5                PUSH  D
066 E7CB CD9DE7            CALL  :E79D      Eval 2nd string expr
067 E7CE F1                POP   PSW
068 E7CF 57                MOV   D,A
069 E7D0 F1                POP   PSW
070 E7D1 C5                PUSH  B          Save it
071 E7D2 44                MOV   B,H
072 E7D3 4D                MOV   C,L
073 E7D4 E1                POP   H
074 E7D5 E3                XTHL             Save program pointer
075 E7D6 C5                PUSH  B          )
076 E7D7 42                MOV   B,D        )
077 E7D8 4B                MOV   C,E        ) Re-arrange registers
078 E7D9 EB                XCHG             )
079 E7DA E1                POP   H          )
080 E7DB FEC0              CPI   :C0        Operator is '+'?
081 E7DD CAEBE7            JZ    :E7EB      Then append 2 strings
082
083                  * If string compare:
084
085 E7E0 CD21D1            CALL  :D121      Compare 2 strings
086 E7E3 CDF8E7            CALL  :E7F8      Returns operands if temp
087 E7E6 C1                POP   B          restore
088 E7E7 5F                MOV   E,A        Opcode in E
089 E7E8 C333E9            JMP   :E933      Evaluate the compare
090
091                  * If operator is '+':
092
093 E7EB E5          ROS10  PUSH  H
094 E7EC CD06D1            CALL  :D106      Make 1 string out of 2
095 E7EF E3                XTHL             Save pntr to result/store
096                                         pntr to operand
097 E7F0 CDF8E7            CALL  :E7F8      Clean up heap
098 E7F3 E1                POP   H          Pntr to result in HL
099 E7F4 C1                POP   B          Program pntr in BC
100 E7F5 1E02              MVI   E,:02      Status: temporary
101 E7F7 C9                RET
102                  *
103                  * CLEAR UP HEAP AFTER STRING OPERATION:
104                  *
105                  * Entry: B,C: Code for 1st resp. 2nd operand.
106                  *             (0=const, 1=var, 2=temp).
107                  *        DE:  Points to 1st operand.
108                  *        HL:  Points to 2nd operand.
109                  * Exit:  DEHL corrupted, AFBC preserved.
110                  *
111 E7F8 F5          DROPS  PUSH  PSW
112 E7F9 79                MOV   A,C        Get code 2nd operand
113 E7FA FE02              CPI   :02        Temporary?
114 E7FC CC87D1            CZ    :D187      Then clear string in heap
115 E7FF EB                XCHG
116 E800 78                MOV   A,B        Get code 1st operand
117 E801 FE02              CPI   :02        Temporary?
118 E803 CC87D1            CZ    :D187      Then clear string in heap
119 E806 F1                POP   PSW
120 E807 C9                RET
121                  *
122                  ************************************
123                  * EVALUATE A NUMERIC EXPRESSION *
124                  ************************************
125                  *
```

```
126                 * Entry: BC: Points to a numeric function argument
127                 *            or a numeric expression in program.
128                 * Exit:  BC updated, AFDEHL preserved.
129                 *        Result in MACC.
130                 *
131 E808 F5          REXNA   PUSH  PSW
132 E809 D5                  PUSH  D
133 E80A E5                  PUSH  H
134 E80B CD19E8              CALL  :E819     Eval arguments in num expr
135 E80E 7C                  MOV   A,H
136 E80F B5                  ORA   L
137 E810 CA15E8              JZ    :E815     Abort if result in MACC
138 E813 E7                  RST   4         Else: copy operand to MACC
139 E814 0C                  DATA  :0C
140 E815 E1          LOE146  POP   H
141 E816 D1                  POP   D
142 E817 F1                  POP   PSW
143 E818 C9                  RET
144                 *
145                 ***********************************************
146                 * EVALUATE ARGUMENTS IN NUMERIC EXPRESSION *
147                 ***********************************************
148                 *
149                 * Checks for constants, functions, variables and
150                 * operators. The right-hand side of the expression
151                 * is therefore evaluated. The value of the variable
152                 * is stored at its varptr location.
153                 *
154                 * Entry: BC:   Points to expression in program.
155                 *        (BC): 1.... Expr begins with operator.
156                 *              01... Variable reference.
157                 *              001.. Function calll.
158                 *              Else  Constant.
159                 *        D<>0: MACC must be preserved.
160                 *
161 E819 1600        REXPN   MVI   D,:00     Set MACC free
162
163                 * Called by lower levels:
164
165 E81B 0A          RXN10   LDAX  B         Get 1st byte
166 E81C 07                  RLC
167 E81D DA50E8              JC    :E850     Jump if expr starts with
168                                          operator
169 E820 07                  RLC
170 E821 DA6BE9              JC    :E96B     Jump if var reference
171 E824 07                  RLC
172 E825 DA30E8              JC    :E830     Jump if function call
173
174                 * If numeric constant:
175
176 E828 03                  INX   B         Past flag byte
177 E829 60                  MOV   H,B       ) HL pnts to constant
178 E82A 69                  MOV   L,C       )
179 E82B 03                  INX   B
180 E82C 03                  INX   B
181 E82D 03                  INX   B
182 E82E 03                  INX   B         Program pntr pnts beyond
183 E82F C9                  RET
184
185                 * If numeric function:
186
187 E830 D5          RFUNN   PUSH  D
```

```
188 E831 7A                 MOV   A,D
189 E832 B7                 ORA   A
190 E833 CA46E8             JZ    :E846       Jump if MACC free
191 E836 CD18C0             CALL  :C018       Save MACC on stack
192 E839 CDD9E9             CALL  :E9D9       Evaluate function call
193                                           result in MACC
194 E83C 212901             LXI   H,:0129     Addr WORKE
195 E83F E7                 RST   4           Copy result to WORKE
196 E840 0F                 DATA  :0F
197 E841 CD1BC0             CALL  :C01B       Restore MACC from stack
198 E844 D1                 POP   D
199 E845 C9                 RET
200 E846 CDD9E9     RFN10   CALL  :E9D9       Evaluate function call,
201                                           result in MACC
202 E849 D1                 POP   D
203 E84A 16FF               MVI   D,:FF       Set MACC to be preserved
204 E84C 210000             LXI   H,:00       Flag 'result in MACC'
205 E84F C9                 RET
206
207                 * If expr begins with numeric operator:
208
209 E850 0A         RONUM   LDAX  B           Get operator
210 E851 E67F               ANI   :7F         Clip operator bit
211 E853 03                 INX   B
212 E854 FE1A               CPI   :1A         Bracket?
213 E856 CA1BE8             JZ    :E81B       Then ignore it
214 E859 15                 DCR   D
215 E85A 14                 INR   D           Check if D<>0
216 E85B C418C0             CNZ   :C018       Then save MACC on stack
217 E85E D5                 PUSH  D
218 E85F 5F                 MOV   E,A         Opcode in E
219 E860 21DCE8             LXI   H,:E8DC
220 E863 E5                 PUSH  H           Returnaddr on stack
221 E864 CD19E8             CALL  :E819       Get 1st operand
222 E867 7B                 MOV   A,E         Opcode in A
223 E868 E61F               ANI   :1F
224 E86A FE1C               CPI   :1C
225 E86C D29FE8             JNC   :E89F       Jump if unitary operation
226
227                 * If boolean operator:
228
229 E86F E5                 PUSH  H           Save pntr to 1st operand
230 E870 CD1BE8             CALL  :E81B       Get 2nd operand
231 E873 7C                 MOV   A,H
232 E874 B5                 ORA   L
233 E875 C27DE8             JNZ   :E87D       Jump if HL pnts to WORKE
234 E878 212901             LXI   H,:0129     Addr WORKE
235 E87B E7                 RST   4           Copy 2nd operand to WORKE
236 E87C 0F                 DATA  :0F
237 E87D E3         RON10   XTHL
238 E87E 7C                 MOV   A,H
239 E87F B5                 ORA   L
240 E880 CA85E8             JZ    :E885       Jump if 1st operand in MACC
241 E883 E7                 RST   4           Else copy it from WORKE
242 E884 0C                 DATA  :0C         to MACC
243 E885 7B         LOE153  MOV   A,E         Get opcode
244 E886 E61F               ANI   :1F
245 E888 FE10               CPI   :10
246 E88A D2C9E8             JNC   :E8C9       If relational operation
247
248                 * If arithmetic operation:
```

```
250 E88D BB                 CMP   E
251 E88E 21FDE8             LXI   H,:E8FD    Addr table INT routines
252 E891 C297E8             JNZ   :E897      Jump if INT
253 E894 21EEE8             LXI   H,:E8EE    Addr table FPT routines
254 E897 1600       RON20   MVI   D,:00      Set MACC free
255 E899 5F                 MOV   E,A        Opcode in E
256 E89A 19                 DAD   D
257 E89B 19                 DAD   D
258 E89C 19                 DAD   D          Find routine in table
259 E89D E3                 XTHL             Addr routine on stack; pntr
260                                          to 2nd operand in HL
261 E89E C9                 RET              Perform routine
262                 *
263                 * If an unitary operation:
264                 *
265                 * Entry: HL: Points to operand (0 if in MACC).
266                 *         E:  Full opcode.
267                 *         A:  Lower 5 bits opcode.
268                 *         Returnaddr on stack (E8DC).
269                 * Exit:   Result in MACC.
270                 *         ABCDEHL preserved
271                 *
272 E89F F5            RON40   PUSH  PSW
273 E8A0 7C                    MOV   A,H
274 E8A1 B5                    ORA   L
275 E8A2 CAA7E8              JZ    :E8A7      If operand in MACC
276 E8A5 E7                  RST   4          Else: operand in MACC
277 E8A6 0C                  DATA  :0C
278 E8A7 F1            LOE156  POP   PSW
279 E8A8 C8                    RZ             Ready if unitary '+'
280 E8A9 BB                    CMP   E        Bits 6,7 opcode 0?
281 E8AA CABEE8              JZ    :E8BE      Then change MACC to INT
282 E8AD FE1E                CPI   :1E        INOT?
283 E8AF DABBE8              JC    :E8BB      Then change sign MACC (INT)
284 E8B2 C2B8E8              JNZ   :E8B8      Then convert MACC to FPT
285 E8B5 E7                  RST   4          Perform INOT
286 E8B6 6C                  DATA  :6C
287 E8B7 C9                  RET
288                 *
289 E8B8 E7            LOE157  RST   4          Convert MACC to FPT
290 E8B9 4B                    DATA  :4B
291 E8BA C9                    RET
292                 *
293 E8BB E7            RON44   RST   4          Change sign MACC (INT)
294 E8BC 60                    DATA  :60
295 E8BD C9                    RET
296                 *
297 E8BE FE1D          RON45   CPI   :1D
298 E8C0 CAC6E8              JZ    :E8C6      Then change sign MACC (FPT)
299 E8C3 E7                  RST   4          Convert MACC to INT
300 E8C4 48                  DATA  :48
301 E8C5 C9                  RET
302                 *
303 E8C6 E7            RON49   RST   4          Change sign MACC (FPT)
304 E8C7 1B                    DATA  :1B
305 E8C8 C9                    RET
306                 *
307                 * If relational numeric operation:
308                 *
309                 * Entry: 1st operand in MACC, 2nd operand on stack.
310                 *         E: Full opcode.
311                 *         A: Lowest 5 bits opcode.
```

```
                        * Exit:  BC preserved, DEHL corrupted.
                        *
E8C9 E1         RON50   POP   H          Get pntr 2nd operand
E8CA BB                 CMP   E
E8CB CAD6E8             JZ    :E8D6      Jump if FPT
E8CE CD15C0             CALL  :C015      Compare 2 INT numbers
E8D1 E1         RON55   POP   H          Kill returnaddr
E8D2 E1                 POP   H          Kill saved DE
E8D3 C333E9             JMP   :E933      Return logical result
E8D6 CD0CC0     RON60   CALL  :C00C      Compare 2 FPT numbers
E8D9 C3D1E8             JMP   :E8D1
                        *
                        * MOVE OPERAND:
                        *
                        * REX.. routines return here after operation.
                        * Moves operand to proper location after computing.
                        *
                        * Entry: DE and returnaddress on stack.
                        * Exit:  ABC preserved.
                        *
E8DC D1         RON30   POP   D
E8DD 15                 DCR   D
E8DE 14                 INR   D          Check D=0 (MACC free)
E8DF 16FF               MVI   D,:FF
E8E1 210000             LXI   H,:0000    Flag 'result in MACC'
E8E4 C8                 RZ               Abort if operand in MACC
E8E5 212901             LXI   H,:0129    Addr WORKE
E8E8 E7                 RST   4          Copy MACC to WORKE
E8E9 0F                 DATA  :0F
E8EA CD1BC0             CALL  :C01B      Restore old MACC from stack
E8ED C9                 RET
                        *
                        * TABLE OF JUMPS TO INT/FPT OPERATOR ROUTINES:
                        *
E8EE E7         ROFTAB  RST   4
E8EF 00                 DATA  :00        MFADD; +
E8F0 C9                 RET
                        *
E8F1 E7                 RST   4
E8F2 03                 DATA  :03        MFSUB; -
E8F3 C9                 RET
                        *
E8F4 E7                 RST   4
E8F5 09                 DATA  :09        MFDIV; /
E8F6 C9                 RET
                        *
E8F7 E7                 RST   4
E8F8 06                 DATA  :06        MFMUL; *
E8F9 C9                 RET
                        *
E8FA E7                 RST   4
E8FB 24                 DATA  :24        MPWR ; ^
E8FC C9                 RET
                        *
E8FD E7         ROITAB  RST   4
E8FE 4E                 DATA  :4E        MIADD; +
E8FF C9                 RET
                        *
E900 E7                 RST   4
E901 51                 DATA  :51        MISUB; -
E902 C9                 RET
                        *
```

```
374 E903 E7                     RST   4
375 E904 57                     DATA  :57       MIDIV; /
376 E905 C9                     RET
377                     *
378 E906 E7                     RST   4
379 E907 54                     DATA  :54       MIMUL; *
380 E908 C9                     RET
381                     *
382 E909 00                     DATA  :00
383 E90A 00                     DATA  :00
384 E90B 00                     DATA  :00
385 E90C 00                     DATA  :00
386 E90D 00                     DATA  :00
387 E90E 00                     DATA  :00
388 E90F 00                     DATA  :00
389 E910 00                     DATA  :00
390 E911 00                     DATA  :00
391 E912 00                     DATA  :00
392 E913 00                     DATA  :00
393 E914 00                     DATA  :00
394 E915 00                     DATA  :00
395 E916 00                     DATA  :00
396 E917 00                     DATA  :00
397                     *
398 E918 E7                     RST   4         MIAND
399 E919 63                     DATA  :63
400 E91A C9                     RET
401                     *
402 E91B E7                     RST   4         MIOR
403 E91C 66                     DATA  :66
404 E91D C9                     RET
405                     *
406 E91E 00                     DATA  :00
407 E91F 00                     DATA  :00
408 E920 00                     DATA  :00
409                     *
410 E921 E7                     RST   4         MIXOR
411 E922 69                     DATA  :69
412 E923 C9                     RET
413                     *
414 E924 E7                     RST   4         MSHL
415 E925 6F                     DATA  :6F
416 E926 C9                     RET
417                     *
418 E927 E7                     RST   4         MSHR
419 E928 72                     DATA  :72
420 E929 C9                     RET
421                     *
422 E92A E7                     RST   4         MIREM
423 E92B 5A                     DATA  :5A
424 E92C C9                     RET
425                     *
426                     *
427                     *
428 E92D                        END

****************************
* S Y M B O L   T A B L E *
****************************

DROPS  E7F8   L0E146 E815   L0E153 E885   L0E156 E8A7
L0E157 E8B8   REXNA  E808   REXPN  E819   REXPS  E79D
```

RFN10	E846	RFUNN	E830	ROFTAB	E8EE	ROITAB	E8FD
RON10	E87D	RON20	E897	RON30	E8DC	RON40	E89F
RON44	E8BB	RON45	E8BE	RON49	E8C6	RON50	E8C9
RON55	E8D1	RON60	E8D6	RONUM	E850	ROS10	E7EB
ROSTR	E7BD	RXN10	E81B	RXS10	E7B3		

```
002                             ORG   :E92D
003                     *
004                     *
005                     *
006                     ***************************
007                     * LENGTH OF BLOCK IN BC *
008                     ***************************
009                     *
010                     * Part of Run 'CLEAR' (D214).
011                     *
012 E92D CD1ADE         MPT45   CALL  :DE1A     Calc. length of block
013 E930 44                     MOV   B,H       )
014 E931 4D                     MOV   C,L       ) Length in BC
015 E932 C9                     RET
016                     *
017                     ************************
018                     * EVALUATE A COMPARE *
019                     ************************
020                     *
021                     * Decodes flags and opcode to a truthtable.
022                     * Following a XFCOMP or a XICOMP by a jump to
023                     * E933 leaves FF (true) or 00 (false) in A as
024                     * result.
025                     *
026                     * Entry: E: Opcode.
027                     *        F: Flags.
028                     * Exit:  BCDEF preserved, HL corrupted.
029                     *        A: Truth value.
030                     *
031 E933 F5             RAREL   PUSH  PSW       Save flags
032 E934 7B                     MOV   A,E       )
033 E935 E60F                   ANI   :0F       ) Calc offset
034 E937 87                     ADD   A         )
035 E938 87                     ADD   A         )
036 E939 2143E9                 LXI   H,:E943   Base addr
037 E93C CD30DE                 CALL  :DE30     Add offset to base
038 E93F F1                     POP   PSW       Restore flags
039 E940 3EFF                   MVI   A,:FF     Init truth value
040 E942 E9                     PCHL            Goto routine
041                     *
042 E943 F0             RAGEQ   RP              FF if MACC >= M
043 E944 C358E9                 JMP   :E958     (S=0)
044                     *
045 E947 FA58E9         RAGT    JM    :E958     FF if MACC > M
046 E94A 00                     NOP             (S=0 and Z=0)
047                     *
048 E94B C0             RANEQ   RNZ             FF if MACC <> M
049 E94C C358E9                 JMP   :E958     (Z=0)
050                     *
051 E94F C8             RALEQ   RZ              FF if MACC <= M
052 E950 00                     NOP             (S=1 or Z=1)
053 E951 00                     NOP
054 E952 00                     NOP
055                     *
056 E953 F8             RALT    RM              FF if MACC < M
057 E954 C358E9                 JMP   :E958     (S=1)
058                     *
059 E957 C8             RAEQ    RZ              FF if MACC = M
060                             (Z=1)
061                     *
062 E958 2F             RAFALSE CMA             00 if condition false
063 E959 C9                     RET
```

```
                        *
                        ****************************
                        * RUN A VARIABLE REFERENCE *
                        ****************************
                        *
                        * Produces a pointer to the value of a variable.
                        * The variable may be simple or subscripted and
                        * of any type. If subscripted, subscripts are
                        * evaluated and checked for range.
                        *
                        * Entry: BC: Points to encoded variable.
                        *        D:  0 if MACC free.
                        * Exit:  BC updated, DE preserved, F corrupted.
                        *        HL: Points to variable storage.
                        *        A:  Type of variable from symbol table.
                        *
E95A D5                 RARRN   PUSH  D
E95B AF                         XRA   A          Array name only
                        *
E95C 1600               RVREN   MVI   D,:00
E95E C33DD7                     JMP   :D73D      Run varptr
                        *
E961 FF                         DATA  :FF
E962 FF                         DATA  :FF
                        *
                        ****************************************
                        * RUN VARPTR (ARRAY WITH ARGUMENTS) *
                        ****************************************
                        *
                        * Runs a varptr with A=FF, D=0.
                        *
                        * Exit: BC updated, DE preserved, MACC corrupted.
                        *       HL: Varptr.
                        *       A:  T/L byte.
                        *
E963 D5                 RVAR    PUSH  D
E964 3EFF                       MVI   A,:FF      Set mask for arrays (not
                                                 name only)
E966 C35CE9                     JMP   :E95C      Run varptr
                        *
E969 FF                         DATA  :FF
E96A FF                         DATA  :FF
                        *
                        ****************************
                        * RUN A VARIABLE POINTER *
                        ****************************
                        *
                        * RVARE: Entry for arrays.
                        * RVR05: 'Normal' entry.
                        *
                        * Entry: BC: Points to var.reference in program.
                        *        A:  Mask.
                        *        D:  0 if MACC free.
                        * Exit:  HL: Varptr (for strings: stringpntr).
                        *        A:  T/L of symtab.
                        *        BC updated, DE preserved.
                        *
E96B 3EFF               RVARE   MVI   A,:FF      Not array name only
E96D F5                 RVR05   PUSH  PSW
E96E D5                         PUSH  D
E96F 0A                         LDAX  B          )
E970 03                         INX   B          )
```

```
126 E971 E63F            ANI   :3F      ) Get offset in symtab
127 E973 57              MOV   D,A       ) in DE
128 E974 0A              LDAX  B         )
129 E975 03              INX   B         )
130 E976 5F              MOV   E,A       )
131 E977 2AA102          LHLD  :02A1     Get start symtab
132 E97A 19              DAD   D         HL pnts to actual addr
133                                      in symtab
134 E97B D1              POP   D
135 E97C F1              POP   PSW       Restore mask
136 E97D A6              ANA   M         And with T/L byte
137 E97E E640            ANI   :40       Bit 6 only
138 E980 7E              MOV   A,M
139 E981 23              INX   H
140 E982 C8              RZ              Abort if simple variable or
141                                      array name only
142
143                * Compute actual position in array:
144
145 E983 15              DCR   D
146 E984 14              INR   D         Check D=0: MACC free
147 E985 C418C0          CNZ   :C018     Save MACC on stack if not
148 E988 D5              PUSH  D
149 E989 F5              PUSH  PSW
150 E98A 5E              MOV   E,M       ) Get pntr from symtab
151 E98B 23              INX   H         ) in DE
152 E98C 56              MOV   D,M       )
153 E98D 23              INX   H
154 E98E 7A              MOV   A,D       ) Check if undimensioned
155 E98F B3              ORA   E         )
156 E990 3E0F      ERRUA MVI   A,:0F     Then run error
157 E992 CAF5D9          JZ    :D9F5     'UNDEFINED ARRAY'
158 E995 D5              PUSH  D
159 E996 210000          LXI   H,:0000   Init index
160 E999 E3              XTHL            Pntr to array in HL
161 E99A 0A              LDAX  B         Get nr of subscripts
162 E99B 03              INX   B
163 E99C 57              MOV   D,A       in D
164 E99D BE              CMP   M         Comp. with nr of dimensions
165 E99E 23              INX   H
166 E99F C229DA          JNZ   :DA29     Run 'SUBSCRIPT ERROR' if
167                                      not identical
168 E9A2 CD4FE7    RVR10 CALL  :E74F     Get variable type in A
169 E9A5 BE        RVR15 CMP   M
170 E9A6 DAB1E9          JC    :E9B1     If value in A is offset
171 E9A9 CAB1E9          JZ    :E9B1
172 E9AC 3E05            MVI   A,:05     If subscript <0 or >FF:
173 E9AE C3F5D9          JMP   :D9F5     Run 'SUBSCRIPT ERROR'
174
175                * Calc reference to Nth array element
176                * via (a1*(d2+1)+a2*(d3+1)+..+aN). aN
177                * is argument, dN is dimension:
178
179 E9B1 E3          RVR20 XTHL            Restore HL=00
180 E9B2 CD30DE           CALL  :DE30     Add offset to index
181 E9B5 15              DCR   D         decr nr of arguments
182 E9B6 E3              XTHL
183 E9B7 23              INX   H
184 E9B8 CAC5E9          JZ    :E9C5     If no more subscipts
185 E9BB 7E              MOV   A,M       Next dimension
186 E9BC 3C              INR   A
187 E9BD E3              XTHL
```

```
188 E9BE CD8FDE             CALL  :DE8F     Multiply index by it
189 E9C1 E3                 XTHL
190 E9C2 C3A2E9             JMP   :E9A2     Process next subscript
191                 *
192 E9C5 EB         RVR30   XCHG
193 E9C6 E1                 POP   H         Get index to array from
194                                         symtab
195 E9C7 F1                 POP   PSW       Get type byte
196 E9C8 F5                 PUSH  PSW
197 E9C9 E630               ANI   :30       Bits 5,6 only
198 E9CB FE20               CPI   :20
199 E9CD 29                 DAD   H         Add offset to element
200 E9CE CAD2E9             JZ    :E9D2
201 E9D1 29                 DAD   H
202 E9D2 19         RVR40   DAD   D         Abs addr of element in HL
203                                         (STR: pntr to string)
204 E9D3 F1                 POP   PSW
205 E9D4 D1                 POP   D
206 E9D5 C41BC0             CNZ   :C01B     Evt retrieve MACC from stack
207 E9D8 C9                 RET
208                 *
209                 ******************************
210                 * EVALUATE A FUNCTION CALL *
211                 ******************************
212                 *
213                 * Finds address of function routine in
214                 * indirection table (#E9F0) and performs the
215                 * function routine.
216                 *
217                 * Entry: BC: Points to function label (#20)
218                 *            in program line.
219                 * Exit:  MACC: Numeric result.
220                 *        BC updated, AFDEHL corrupted.
221                 *
222 E9D9 03         RFUN    INX   B         Pnts past label
223 E9DA 0A                 LDAX  B         Get function code
224 E9DB 03                 INX   B
225 E9DC 6F                 MOV   L,A       Code in HL
226 E9DD 2600               MVI   H,:00
227 E9DF 11F0E9             LXI   D,:E9F0   Get startaddr. table
228 E9E2 29                 DAD   H         Code *2
229 E9E3 19                 DAD   D         Add offset to startaddr
230 E9E4 5E                 MOV   E,M       lobyte addr in E
231 E9E5 23                 INX   H
232 E9E6 7E                 MOV   A,M       hibyte addr in A
233 E9E7 F680               ORI   :80       Set msb = 1
234 E9E9 57                 MOV   D,A       hibyte in D
235 E9EA BE                 CMP   M         Was it already E...?
236 E9EB D5                 PUSH  D         Addr function on stack
237 E9EC CC08E8             CZ    :E808     Then evaluate 1st num
238                                         expr, result in MACC
239 E9EF C9                 RET             Go to function routine
240                 *
241                 ******************************
242                 * FUNCTION INDIRECTION TABLE *
243                 ******************************
244                 *
245                 * Startaddress table is E9F0. The function code
246                 * is given between brackets.
247                 * If the msb of the address is set, the first
248                 * argument is a numeric one.
```

```
250                     * In the textbuffer, the Basic fuctions are
251                     * indicated by 20 xx, (xx is function code).
252                     *
253 E9F0 50EA        FUNIT   DBL   :EA50      (00)  ABS
254 E9F2 53EA                DBL   :EA53      (01)  ALOG
255 E9F4 CB6A                DBL   :6ACB      (02)  ASC
256 E9F6 D26A                DBL   :6AD2      (03)  CHR$
257 E9F8 B86A                DBL   :6AB8      (04)  CURY
258 E9FA BE6A                DBL   :6ABE      (05)  CURX
259 E9FC 56EA                DBL   :EA56      (06)  EXP
260 E9FE 59EA                DBL   :EA59      (07)  FRAC
261 EA00 436B                DBL   :6B43      (08)  FRE
262 EA02 5CEB                DBL   :EB5C      (09)  FREQ
263 EA04 796B                DBL   :6B79      (0A)  GETC
264 EA06 836A                DBL   :6A83      (0B)  HEX$
265 EA08 82EB                DBL   :EB82      (0C)  INP
266 EA0A 8DEB                DBL   :EB8D      (0D)  INT
267 EA0C E26A                DBL   :6AE2      (0E)  LEFT$
268 EA0E C46A                DBL   :6AC4      (0F)  LEN
269 EA10 A16B                DBL   :6BA1      (10)  VARPTR
270 EA12 5CEA                DBL   :EA5C      (11)  LOG
271 EA14 5FEA                DBL   :EA5F      (12)  LOGT
272 EA16 A76B                DBL   :6BA7      (13)  XMAX
273 EA18 AE6B                DBL   :6BAE      (14)  YMAX
274 EA1A 0E6B                DBL   :6B0E      (15)  MID$
275 EA1C C16B                DBL   :6BC1      (16)  PDL
276 EA1E 166C                DBL   :6C16      (17)  PEEK
277 EA20 1D6C                DBL   :6C1D      (18)  PI
278 EA22 FF6A                DBL   :6AFF      (19)  RIGHT$
279 EA24 27EC                DBL   :EC27      (1A)  RND
280 EA26 9D6C                DBL   :6C9D      (1B)  SCRN
281 EA28 7BEC                DBL   :EC7B      (1C)  SGN
282 EA2A 8C6A                DBL   :6A8C      (1D)  SPC
283 EA2C 62EA                DBL   :EA62      (1E)  SQR
284 EA2E 77EA                DBL   :EA77      (1F)  STR$
285 EA30 A26A                DBL   :6AA2      (20)  TAB
286 EA32 256B                DBL   :6B25      (21)  VAL
287 EA34 65EA                DBL   :EA65      (22)  SIN
288 EA36 68EA                DBL   :EA68      (23)  COS
289 EA38 6BEA                DBL   :EA6B      (24)  TAN
290 EA3A 6EEA                DBL   :EA6E      (25)  ASIN
291 EA3C 71EA                DBL   :EA71      (26)  ACOS
292 EA3E 74EA                DBL   :EA74      (27)  ATN
293                     *
294                     *
295                     *
296 EA40                    END
```

```
****************************
* S Y M B O L   T A B L E *
****************************

ERRUA   E990    FUNIT   E9F0    MPT45   E92D    RARRN   E95A
RFALSE  E958    RFUN    E9D9    ROEQ    E957    ROGEQ   E943
ROGT    E947    ROLEQ   E94F    ROLT    E953    RONEQ   E94B
OREL    E933    RVAR    E963    RVARE   E96B    RVR05   E96D
RVR10   E9A2    RVR15   E9A5    RVR20   E9B1    RVR30   E9C5
RVR40   E9D2    RVREN   E95C
```

```
002                             ORG   :EA40
003                     *
004                     *
005                     *
006                     ******************************
007                     * part of RUN TALK (0E67B) *
008                     ******************************
009                     *
010                     * Sets oscillator volumes.
011                     *
012                     * Entry: A:  New volume.
013                     *        HL: Address POROM/POR1M.
014                     *
015 EA40 77             MPT48   MOV   M,A       Update POROM/POR1M
016 EA41 1170FA                 LXI   D,:FA70
017 EA44 19                     DAD   D         HL= POR0/POR1
018 EA45 77                     MOV   M,A       Update osc.volume
019 EA46 E1                     POP   H         Get parameter pntr
020 EA47 23             RTK55   INX   H         Pnts to next code
021 EA48 C367CD                 JMP   :CD67     Handle next code
022                     *
023 EA4B FF                     DATA  :FF
024 EA4C FF                     DATA  :FF
025 EA4D FF                     DATA  :FF
026 EA4E FF                     DATA  :FF
027 EA4F FF                     DATA  :FF
028                     *
029                     ****************************
030                     * RUN basicfunction ABS *
031                     ****************************
032                     *
033 EA50 E7             RABS    RST   4         MFABS
034 EA51 18                     DATA  :18
035 EA52 C9                     RET
036                     *
037                     ****************************
038                     * RUN basicfunction ALOG *
039                     ****************************
040                     *
041 EA53 E7             RALOG   RST   4         MALOG
042 EA54 30                     DATA  :30
043 EA55 C9                     RET
044                     *
045                     ****************************
046                     * RUN basicfunction EXP *
047                     ****************************
048                     *
049 EA56 E7             REXP    RST   4         MEXP
050 EA57 2A                     DATA  :2A
051 EA58 C9                     RET
052                     *
053                     ****************************
054                     * RUN basicfunction FRAC *
055                     ****************************
056                     *
057 EA59 E7             RFRAC   RST   4         MFRAC
058 EA5A 21                     DATA  :21
059 EA5B C9                     RET
060                     *
061                     ****************************
062                     * RUN basicfunction LOG *
063                     ****************************
```

```
064                     *
065 EA5C E7             RLOG    RST   4          MLOG
066 EA5D 27                     DATA  :27
067 EA5E C9                     RET
068                     *
069                     ****************************
070                     * RUN basicfunction LOGT *
071                     ****************************
072                     *
073 EA5F E7             RLOGT   RST   4          MLOGT
074 EA60 2D                     DATA  :2D
075 EA61 C9                     RET
076                     *
077                     ****************************
078                     * RUN basicfunction SQR *
079                     ****************************
080                     *
081 EA62 E7             RSQR    RST   4          MSQR
082 EA63 33                     DATA  :33
083 EA64 C9                     RET
084                     *
085                     ****************************
086                     * RUN basicfunction SIN *
087                     ****************************
088                     *
089 EA65 E7             RSIN    RST   4          MSIN
090 EA66 36                     DATA  :36
091 EA67 C9                     RET
092                     *
093                     ****************************
094                     * RUN basicfunction COS *
095                     ****************************
096                     *
097 EA68 E7             RCOS    RST   4          MCOS
098 EA69 39                     DATA  :39
099 EA6A C9                     RET
100                     *
101                     ****************************
102                     * RUN basicfunction TAN *
103                     ****************************
104                     *
105 EA6B E7             RTAN    RST   4          MTAN
106 EA6C 3C                     DATA  :3C
107 EA6D C9                     RET
108                     *
109                     ****************************
110                     * RUN basicfunction ASIN *
111                     ****************************
112                     *
113 EA6E E7             RASIN   RST   4          MASIN
114 EA6F 3F                     DATA  :3F
115 EA70 C9                     RET
116                     *
117                     ****************************
118                     * RUN basicfunction ACOS *
119                     ****************************
120                     *
121 EA71 E7             RACOS   RST   4          MACOS
122 EA72 42                     DATA  :42
123 EA73 C9                     RET
124                     *
125                     *
```

```
126                     ****************************
127                     * RUN basicfunction ATAN *
128                     ****************************
129                     *
130 EA74 E7             RATN    RST   4          MATAN
131 EA75 45                     DATA  :45
132 EA76 C9                     RET
133                     *
134                     ****************************
135                     * RUN basicfunction STR$ *
136                     ****************************
137                     *
138                     * Converts a FPT number into a string.
139                     *
140 EA77 CD9BCE         RSTR    CALL  :CE9B      Convert MACC for FPT output
141                                              string in DECBUF
142 EA7A 00                     NOP
143 EA7B 00                     NOP
144 EA7C 00                     NOP
145 EA7D 2A33C0         RST20   LHLD  :C033      Get addr DECBUF
146 EA80 1E01                   MVI   E,:01      Pretend it is a variable
147 EA82 C9                     RET
148                     *
149                     ****************************
150                     * RUN basicfunction HEX$ *
151                     ****************************
152                     *
153                     * Converts a INT number into an equivalent string.
154                     *
155 EA83 CD08E8         RHEX    CALL  :E808      Eval expr, result in MACC
156 EA86 CD2DC0                 CALL  :C02D      Conv. MACC to HEX for output
157 EA89 C37DEA                 JMP   :EA7D      Get addr DECBUF, pretend it
158                                              is a variable
159                     *
160                     ****************************
161                     * RUN basicfunction SPC *
162                     ****************************
163                     *
164                     * Returns a string of a number of spaces.
165                     * From SPC10 used by TAB if DOUTC<>0.
166                     *
167 EA8C CD1DE7         RSPC    CALL  :E71D      Get nr of spaces in A
168
169                     * Entry from RTAB:
170
171 EA8F CD8BD1         SPC10   CALL  :D18B      Get place in heap for string
172                                              of spaces
173 EA92 E5                     PUSH  H          Save pntr to heap
174 EA93 B7             SPC20   ORA   A
175 EA94 CA9EEA                 JZ    :EA9E      Jump if ready
176 EA97 23                     INX   H
177 EA98 3620                   MVI   M,:20      Space into heap
178 EA9A 3D                     DCR   A
179 EA9B C393EA                 JMP   :EA93      Next space
180 EA9E E1             SPC30   POP   H          HL pnts to start string
181 EA9F C3DFEA                 JMP   :EADF      Pretend it is a temp string
182                     *
183                     ****************************
184                     * RUN basicfunction TAB *
185                     ****************************
186                     *
                          Returns a string of spaces to move cursor to a
```

```
188                     * given character position (only to the right).
189                     * Works only if output switch DOUTC=0, else it
190                     * returns one space only.
191                     *
192 EAA2 CD60CE      RTAB    CALL  :CE60     Get nr of tabs in L,
193                                           DOUTC in A
194 EAA5 B7                  ORA   A         Check output direction
195 EAA6 3E01                MVI   A,:01     Init 1 space
196 EAA8 C28FEA              JNZ   :EA8F     Jump if DOUTC<>0
197 EAAB 7D                  MOV   A,L       Get nr of tabs
198 EAAC 00                  NOP
199 EAAD 00                  NOP
200 EAAE EF                  RST   5         Ask cursor pos and size
201 EAAF 0C                  DATA  :0C       char screen
202 EAB0 95                  SUB   L         Calc nr of spaces reqd
203 EAB1 D28FEA              JNC   :EA8F     Run SPC if not past tab pos
204 EAB4 AF                  XRA   A         If past TAB:
205 EAB5 C38FEA              JMP   :EA8F     Run SPC for no spaces
206                     *
207                     ***************************
208                     * RUN basicfunction CURX *
209                     ***************************
210                     *
211 EAB8 EF          RCURX   RST   5         Ask cursor pos and size
212 EAB9 0C                  DATA  :0C       char screen
213 EABA 7D                  MOV   A,L       X-coord in A
214 EABB C37CEB              JMP   :EB7C     and into MACC
215                     *
216                     ***************************
217                     * RUN basicfunction CURY *
218                     ***************************
219                     *
220 EABE EF          RCURY   RST   5         Ask cursor pos and size
221 EABF 0C                  DATA  :0C       char screen
222 EAC0 7C                  MOV   A,H       Y-coord in A
223 EAC1 C37CEB              JMP   :EB7C     and in MACC
224                     *
225                     ***************************
226                     * RUN basicfunction LEN *
227                     ***************************
228                     *
229                     * Given a string, returns length of the string.
230                     *
231 EAC4 CD91E7      RLEN    CALL  :E791     Eval string expr
232 EAC7 7E          RLE10   MOV   A,M       Get length in A
233 EAC8 C37CEB              JMP   :EB7C     and into MACC
234                     *
235                     ***************************
236                     * RUN basicfunction ASC *
237                     ***************************
238                     *
239                     * Given a string, returns ASCII value of 1st char.
240                     *
241 EACB CD91E7      RASC    CALL  :E791     Eval string expr
242 EACE 7E                  MOV   A,M       Get length in A
243 EACF C37ECF              JMP   :CF7E     Check if length is 0; get
244                                           1st char in MACC if not
245                     *
246                     ***************************
247                     * RUN basicfunction CHR$ *
248                     ***************************
249                     *
```

```
EAD2 CD1DE7     RCHR    CALL  :E71D     Get argument value in A
EAD5 F5                 PUSH  PSW       Save it
EAD6 3E01               MVI   A,:01
EAD8 CD8BD1             CALL  :D18B     Find place in heap for
                                        a 1-byte string
EADB F1                 POP   PSW       Get argument
EADC 23                 INX   H
EADD 77                 MOV   M,A       Store it in Heap
EADE 2B                 DCX   H         Pnts to length byte

                * Entry from 'SPC':

EADF 1E02       RCR10   MVI   E,:02     Status: temporary
EAE1 C9                 RET
                *
                ****************************
                * RUN basicfunction LEFT$ *
                ****************************
                *
                * Given a string, returns a number of characters
                * from the left end.
                *
EAE2 CD9DE7     RLEFT   CALL  :E79D     Eval string expr
EAE5 E5                 PUSH  H         Save string pntr
EAE6 D5                 PUSH  D
EAE7 CD1DEB             CALL  :EB1D     Reqd length in A
EAEA 1600               MVI   D,:00
EAEC 5F         RLF10   MOV   E,A       Length in DE
EAED CD4FD1     RLF20   CALL  :D14F     Extract substring
EAF0 D215DA             JNC   :DA15     Evt. run error 'NUMBER OUT
                                        OF RANGE'
EAF3 D1                 POP   D
EAF4 E3                 XTHL
EAF5 7B                 MOV   A,E       Get status
EAF6 FE02               CPI   :02       Temporary?
EAF8 CC87D1             CZ    :D187     Then clear heap entry
EAFB E1                 POP   H
EAFC 1E02               MVI   E,:02     Status temporary
EAFE C9                 RET
                *
                ****************************
                * RUN basicfunction RIGHT$ *
                ****************************
                *
                * Extracts a number of characters from the
                * right end of a given string.
                *
EAFF CD9DE7     RRIGHT  CALL  :E79D     Eval string expr
EB02 E5                 PUSH  H         Save string pntr
EB03 D5                 PUSH  D
EB04 CD1DEB             CALL  :EB1D     Get length substring
EB07 5F                 MOV   E,A       in E
EB08 7E                 MOV   A,M       Get total string length
EB09 93                 SUB   E
EB0A 57                 MOV   D,A       Startposition in D
EB0B C3EDEA             JMP   :EAED     Extract substring
                *
                ***************************
                * RUN basicfunction MID$ *
                ***************************
                *
EB0E CD9DE7     RMID    CALL  :E79D     Eval string expr
```

```
312 EB11 E5              PUSH  H        Save string pntr
313 EB12 D5              PUSH  D
314 EB13 CD1DEB          CALL  :EB1D    Get startposition
315 EB16 57              MOV   D,A      in D
316 EB17 CD1DE7          CALL  :E71D    Get length in A
317 EB1A C3ECEA          JMP   :EAEC    Extract substring
318                  *
319                  ********************************
320                  * GET VALUE OF ARGUMENT IN A *
321                  ********************************
322                  *
323                  * Exit: DEHL preserved.
324                  *
325 EB1D E5          REXIK   PUSH  H
326 EB1E D5                  PUSH  D
327 EB1F CD1DE7              CALL  :E71D    Get value of argument in A
328 EB22 D1                  POP   D
329 EB23 E1                  POP   H
330 EB24 C9                  RET
331                  *
332                  ***************************
333                  * RUN basicfunction VAL *
334                  ***************************
335                  *
336                  * Takes a string and converts it to a FPT number.
337                  *
338 EB25 CD91E7      RVAL    CALL  :E791    Eval string expr
339 EB28 C5                  PUSH  B
340 EB29 7E          SUEPT   MOV   A,M      Length of string in A
341 EB2A 323401              STA   :0134    and in EFECT
342 EB2D 23                  INX   H        HL pnts to 1st string byte
343 EB2E 223201              SHLD  :0132    Addr into EFEPT
344 EB31 0E00                MVI   C,:00    Char count
345 EB33 213501              LXI   H,:0135
346 EB36 3601                MVI   M,:01    Input from string
347 EB38 CD1EC0              CALL  :C01E    encode FPT number into MACC
348 EB3B 35                  DCR   M        Input from keyboard
349 EB3C 3E0A                MVI   A,:0A    If over/underflow: run
350 EB3E D2F5D9              JNC   :D9F5    error 'INVALID NUMBER'
351 EB41 C1                  POP   B
352 EB42 C9                  RET
353                  *
354                  ***************************
355                  * RUN basicfunction FRE *
356                  ***************************
357                  *
358                  * Returns a INT given size of free RAM space.
359                  * Result in MACC.
360                  * FR2BY: Also used to copy HL into MACC.
361                  *
362                  * Exit: BC preserved, AFDEHL corrupted.
363                  *
364 EB43 CD51EB      RFRE    CALL  :EB51    Calc free RAM space in HL
365 EB46 AF          FR2BY   XRA   A
366 EB47 C5                  PUSH  B
367 EB48 D5                  PUSH  D
368 EB49 4C                  MOV   C,H      ) Free space in CD
369 EB4A 55                  MOV   D,L      )
370 EB4B 47                  MOV   B,A      A,B=0
371 EB4C E7                  RST   4        Copy reg A,B,C,D into MACC
372 EB4D 12                  DATA  :12
373 EB4E D1                  POP   D
```

```
374 EB4F C1                 POP   B
375 EB50 C9                 RET
376                 *
377                 ****************************
378                 * CALCULATE FREE RAM SPACE *
379                 ****************************
380                 *
381                 * Exit: HL: Free RAM space.
382                 *       DE: STBUSE.
383                 *       ABC preserved, F corrupted.
384                 *
385 EB51 2AA302    SIZE    LHLD  :02A3     Get end symtab
386 EB54 EB               XCHG              in DE
387 EB55 2AA502           LHLD  :02A5     Get bottom screen RAM
388 EB58 CD1ADE          CALL  :DE1A     Calc. free space in HL
389 EB5B C9              RET
390                 *
391                 ***************************
392                 * RUN basicfunction FREQ *
393                 ***************************
394                 *
395                 * Given a frequency in Hz, returns a period in
396                 * 'oscillator cycles' (INT).
397                 *
398                 * Entry: MACC: Value for freq.
399                 * Exit:  BC preserved, AFDEHL corrupted.
400                 *
401 EB5C 212901    RFREQ   LXI   H,:0129   Startaddr scratch area for
402                                         expression evaluation
403 EB5F E7                 RST   4         Copy reqd freq to scratch
404 EB60 0F                 DATA  :0F       area
405 EB61 E5                 PUSH  H         Save startaddr scratch area
406 EB62 21EDD0             LXI   H,:D0ED   Addr sound constant
407 EB65 E7                 RST   4         Sound constant into MACC
408 EB66 0C                 DATA  :0C
409 EB67 E1                 POP   H         Get start scratch area
410 EB68 E7                 RST   4         Calc sound const/reqd freq
411 EB69 09                 DATA  :09
412 EB6A E7                 RST   4         Change it to INT
413 EB6B 48                 DATA  :48
414 EB6C C5                 PUSH  B
415 EB6D E7                 RST   4         Copy result to reg A,B,C,D
416 EB6E 15                 DATA  :15
417 EB6F B0                 ORA   B         > 64K ? Then run error
418 EB70 C215DA             JNZ   :DA15     'NUMBER OUT OF RANGE'
419 EB73 C1                 POP   B
420 EB74 C9                 RET
421                 *
422                 **********************
423                 * DATA - (not used) *
424                 **********************
425                 *
426 EB75 15         L0E235  DATA  :15       Sound constant
427 EB76 F4                 DATA  :F4
428 EB77 24                 DATA  :24
429 EB78 00                 DATA  :00
430                 *
431                 ***************************
432                 * RUN basicfunction GETC *
433                 ***************************
434                 *
435                 * Gets one character from keyboard. Returns its
```

```
                        * ASCII value in MACC; 0 if no inputs.
                        * FR1BY: Also used to copy 1 byte into MACC.
                        *
EB79 CDBBD6             RGETC   CALL  :D6BB     Scan keyboard, result in A
EB7C 6F                 FR1BY   MOV   L,A       Result in L
EB7D 2600                       MVI   H,:00
EB7F C346EB                     JMP   :EB46     Result into MACC
                        *
                        ***************************
                        * RUN basicfunction INP *
                        ***************************
                        *
                        * Reads a byte from a Real World address (DCE-bus).
                        *
EB82 CD1DE7             RINP    CALL  :E71D     Get RW addr in A
EB85 57                         MOV   D,A       and in D
EB86 CDE0D8                     CALL  :D8E0     Get input from DCE-bus
EB89 7B                         MOV   A,E       Result in A
EB8A C37CEB                     JMP   :EB7C     Result into MACC
                        *
                        ***************************
                        * RUN basicfunction INT *
                        ***************************
                        *
                        * Returns a integer FPT value, just less than the
                        * FPT argument given.
                        * REMARK: Routine is wrong if -1 < nr < 0. Then
                        *         the result is -1 !
                        *
EB8D C5                 RINT    PUSH  B
EB8E E7                         RST   4         Copy MACC to reg A,B,C,D
EB8F 15                         DATA  :15
EB90 C1                         POP   B
EB91 E7                         RST   4         Change MACC to INT, and then
EB92 1E                         DATA  :1E       to FPT
EB93 21F1D0                     LXI   H,:D0F1   Addr FPT(-1)
EB96 B7                         ORA   A
EB97 F29CEB                     JP    :EB9C     Abort if positive
EB9A E7                         RST   4         Add -1 if MACC negative
EB9B 00                         DATA  :00
EB9C C9                 LOE239  RET
                        *
                        ***********************
                        * DATA - (not used) *
                        ***********************
                        *
EB9D 81                 LOE240  DATA  :81       FPT (-1)
EB9E 80                         DATA  :80
EB9F 00                         DATA  :00
EBA0 00                         DATA  :00
                        *
                        *****************************
                        * RUN basicfunction VARPTR *
                        *****************************
                        *
EBA1 CD63E9             RVPT    CALL  :E963     Get varptr in HL, T/L in A
EBA4 C346EB                     JMP   :EB46     Varptr into MACC
                        *
                        ***************************
                        * RUN basicfunction XMAX *
                        ***************************
                        *
```

```
498 EBA7 CDB4EB     RXMAX   CALL  :EBB4     Get max Y,X-coord graph area
499 EBAA EB                 XCHG            Max X-coord in HL
500 EBAB C346EB             JMP   :EB46     and into MACC
501                 *
502                 ***************************
503                 * RUN basicfunction YMAX *
504                 ***************************
505                 *
506 EBAE CDB4EB     RYMAX   CALL  :EBB4     Get max Y,X-coord graph area
507 EBB1 C37CEB             JMP   :EB7C     Max Y-coord into MACC
508                 *
509                 ******************************************
510                 * GET MAX. Y,X-COORDINATES GRAPHICS AREA *
511                 ******************************************
512                 *
513                 * Exit: DE: Max. X-coordinate.
514                 *       A:  Max. Y-coordinate.
515                 *       BC preserved.
516                 *
517 EBB4 210000     L0E245  LXI   H,:0000   ) Coord dot 0,0
518 EBB7 C5                 PUSH  B         )
519 EBB8 4C                 MOV   C,H       )
520 EBB9 EF                 RST   5         Ask colour of point and
521 EBBA 27                 DATA  :27       size graphics screen
522 EBBB DA02E6             JC    :E602     Evt run screen error
523 EBBE 78                 MOV   A,B       Max Y-coord in A
524 EBBF C1                 POP   B
525 EBC0 C9                 RET
526                 *
527                 ***************************
528                 * RUN basicfunction PDL *
529                 ***************************
530                 *
531                 * A given paddle channel is enabled. Counter 0 is
532                 * set to FFFF. The counter is read over and over
533                 * until it is counted out.
534                 *
535                 * Exit: BC updated, AFDEHL corrupted.
536                 *
537 EBC1 3E05       RPDL    MVI   A,:05
538 EBC3 CD43E7             CALL  :E743     Get paddle select (0-5)
539 EBC6 57                 MOV   D,A       into D
540 EBC7 3A4000             LDA   :0040     Get POROM
541 EBCA E6F8               ANI   :F8       ROM/cass.select only
542 EBCC B2                 ORA   D         OR with paddle select
543 EBCD F608               ORI   :08       Paddle enable
544 EBCF CD08D8             CALL  :D808     Load PORO/POROM
545 EBD2 C5                 PUSH  B
546 EBD3 3E30               MVI   A,:30
547 EBD5 0106FC             LXI   B,:FC06   Addr 8253 cmd word
548 EBD8 02                 STAX  B         Select ch.0, mode 0, 2 byte
549 EBD9 21FFFF             LXI   H,:FFFF
550 EBDC 2200FC             SHLD  :FC00     Load counter ch.0
551 EBDF 3A01FD             LDA   :FD01     Get pdl timer trig impulse
552                                         (Useless: A is cleared in
553                                         0EBE3)
554 EBE2 EB         PDL10   XCHG            DE = FFFF
555 EBE3 3E00               MVI   A,:00
556 EBE5 02                 STAX  B         (FC06)=00: counter 0, latch
557                                         operation
558 EBE6 2A00FC             LHLD  :FC00     Get contents counter 0
559 EBE9 CD14DE             CALL  :DE14     Compare HL-DE
```

```
560 EBEC DAE2EB             JC    :EBE2     Again if DE > HL
561 EBEF CD26DE             CALL  :DE26     HL = 2-compl. of HL
562 EBF2 11CEFF             LXI   D,:FFCE   ) Substract 49
563 EBF5 19                 DAD   D         )
564 EBF6 DAFCEB             JC    :EBFC     If result negative
565 EBF9 210000             LXI   H,:0000
566 EBFC 7C          PDL20  MOV   A,H
567 EBFD B7                 ORA   A
568 EBFE CA06EC             JZ    :EC06
569 EC01 2EFF               MVI   L,:FF
570 EC03 00                 NOP
571 EC04 00                 NOP
572 EC05 00                 NOP
573 EC06 3E36        PDL30  MVI   A,:36
574 EC08 02                 STAX  B         (FC06)=#36: Chan 0, mode 3
575 EC09 3A4000             LDA   :0040     Get PORO/POROM
576 EC0C E6F0               ANI   :F0       Disable paddle operation
577 EC0E CD06D8             CALL  :D806     Load PORO/POROM
578 EC11 C1                 POP   B
579 EC12 7D                 MOV   A,L       A=0 if result negative,
580                                         else FF
581 EC13 C37CEB             JMP   :EB7C     Move A into MACC
582                  *
583                  *
584                  *
585 EC16                    END
```

```
****************************
* S Y M B O L   T A B L E *
****************************
```

FR1BY	EB7C	FR2BY	EB46	L0E235	EB75	L0E239	EB9C
L0E240	EB9D	L0E245	EBB4	MPT48	EA40	PDL10	EBE2
PDL20	EBFC	PDL30	EC06	RABS	EA50	RACOS	EA71
RALOG	EA53	RASC	EACB	RASIN	EA6E	RATN	EA74
RCHR	EAD2	RCOS	EA68	RCR10	EADF	RCURX	EAB8
RCURY	EABE	REXIK	EB1D	REXP	EA56	RFRAC	EA59
RFRE	EB43	RFREQ	EB5C	RGETC	EB79	RHEX	EA83
RINP	EB82	RINT	EB8D	RLE10	EAC7	RLEFT	EAE2
RLEN	EAC4	RLF10	EAEC	RLF20	EAED	RLOG	EA5C
RLOGT	EA5F	RMID	EB0E	RPDL	EBC1	RRIGHT	EAFF
RSIN	EA65	RSPC	EA8C	RSQR	EA62	RST20	EA7D
RSTR	EA77	RTAB	EAA2	RTAN	EA6B	RTK55	EA47
RVAL	EB25	RVPT	EBA1	RXMAX	EBA7	RYMAX	EBAE
SIZE	EB51	SPC10	EA8F	SPC20	EA93	SPC30	EA9E
SUEPT	EB29						

```
002                             ORG   :EC16
003                     *
004                     *
005                     *
006                     ****************************
007                     * RUN basicfunction PEEK *
008                     ****************************
009                     *
010                     * Returns the contents of a memory location
011                     * with an address given as INT argument.
012                     *
013 EC16 CDF8E6         RPEEK   CALL  :E6F8     Get addr in HL
014 EC19 7E                     MOV   A,M       Get its contents
015 EC1A C37CEB                 JMP   :EB7C     Move it into MACC
016                     *
017                     ************************
018                     * RUN basicfunction PI *
019                     ************************
020                     *
021                     * Returns a value of 3.14159.
022                     *
023 EC1D 21F5D0         RPI     LXI   H,:D0F5   Addr FPT (PI)
024 EC20 E7                     RST   4         FPT (PI) into MACC
025 EC21 0C                     DATA  :0C
026 EC22 C9                     RET
027                     *
028                     **********************
029                     * DATA - (not used) *
030                     **********************
031                     *
032 EC23 02             L0E252  DATA  :02       FPT (PI)
033 EC24 C9                     DATA  :C9
034 EC25 0F                     DATA  :0F
035 EC26 DB                     DATA  :DB
036                     *
037                     ****************************
038                     * RUN basicfunction RND *
039                     ****************************
040                     *
041                     * Returns a random or pseudo-random number.
042                     * If argument > 0: Returns a pseudo-random number
043                     *                  in the range 0 <= R < argument.
044                     * If argument = 0: Returns a hardware random number
045                     *                  0 < R < 1.
046                     * If argument < 0: Number replaces the kernel for
047                     *                  calculating further pseudo-
048                     *                  random numbers.
049                     *
050 EC27 CD8AEC         RRND    CALL  :EC8A     Test if arg is 0
051 EC2A CA0CE4                 JZ    :E40C     Then hardware random
052 EC2D 212D01                 LXI   H,:012D   Addr randdom number kernel
053 EC30 F235EC                 JP    :EC35     If pseudo random number
054 EC33 E7                     RST   4         Copy MACC to kernel
055 EC34 0F                     DATA  :0F
056 EC35 EB                     XCHG
057 EC36 212901                 LXI   H,:0129   Addr scratch area WORKE
058 EC39 E7                     RST   4         Copy MACC to WORKE
059 EC3A 0F                     DATA  :0F
060 EC3B E5                     PUSH  H         Saveaddr WORKE
061 EC3C EB                     XCHG
062 EC3D E5                     PUSH  H         Save addr RNUM
                                MVI   M,:01     Limit range 1-2
```

```
064 EC40 E7                 RST   4          Copy last nr from RNUM
065 EC41 0C                 DATA  :0C        into MACC
066 EC42 1605               MVI   D,:05
067 EC44 21A8C6     RRD10   LXI   H,:C6A8    Addr RNDA
068 EC47 E7                 RST   4          Multiply R0*RNDA
069 EC48 54                 DATA  :54
070 EC49 21ACC6             LXI   H,:C6AC    Addr RNDB
071 EC4C E7                 RST   4          Add RNDB to R0*RNDA
072 EC4D 4E                 DATA  :4E
073 EC4E 00                 NOP
074 EC4F 00                 NOP
075 EC50 00                 NOP
076 EC51 00                 NOP
077 EC52 00                 NOP
078 EC53 21FDD0             LXI   H,:D0FD    Addr AND mask
079 EC56 E7                 RST   4          IAND: pick out mantissa
080 EC57 63                 DATA  :63
081 EC58 21B0C6             LXI   H,:C6B0    Addr OR mask
082 EC5B E7                 RST   4          IOR: set mantissa top
083 EC5C 66                 DATA  :66        bit, + range 1-2
084 EC5D 15                 DCR   D
085 EC5E C244EC             JNZ   :EC44      Again if D<>0
086 EC61 E1                 POP   H          Get addr RNUM
087 EC62 E7                 RST   4          Copy MACC to RNUM
088 EC63 0F                 DATA  :0F
089 EC64 21F1D0             LXI   H,:D0F1    Addr FPT (-1)
090 EC67 E7                 RST   4          Add -1 to MACC (range
091 EC68 00                 DATA  :00        0-1)
092 EC69 E1                 POP   H          Get addr WORKE
093 EC6A E7                 RST   4          Frig range: multiply
094 EC6B 06                 DATA  :06        MACC*(WORKE)
095 EC6C C9                 RET
096                 *
097                 ****************************
098                 * part of RUN TALK (CD64) *
099                 ****************************
100                 *
101                 * Entry: Z=1: Code = 0C (delay).
102                 *          Z=0: Code = 0D (ML call)
103                 *
104 EC6D 5E         MPT46   MOV   E,M        )
105 EC6E 23                 INX   H          ) Wait-time/ML address
106 EC6F 56                 MOV   D,M        ) in HL
107 EC70 23                 INX   H          )
108 EC71 EB                 XCHG             )
109 EC72 CCCCDA             CZ    :DACC      If to be waited
110 EC75 C4A9C8             CNZ   :C8A9      Else: Run ML routine
111 EC78 C367CD             JMP   :CD67      Return: Handle next code
112                 *
113                 ************************
114                 * RUN basicfunction SGN *
115                 ************************
116                 *
117                 * Takes a FPT value and returns:
118                 *                +1  if value is positive.
119                 *                 0  if value is zero.
120                 *                -1  if value is negative.
121                 *
122 EC7B CD8AEC     RSGN    CALL  :EC8A      Test if variable is zero
123 EC7E C8                 RZ               Then ready
124 EC7F 21F1D0             LXI   H,:D0F1    Addr FPT(-1)
125 EC82 E7                 RST   4          Copy -1 into MACC
```

```
126 EC83 0C                 DATA  :0C
127 EC84 FA89EC             JM    :EC89     Ready if already negative
128 EC87 E7                 RST   4         Else change sign MACC
129 EC88 1B                 DATA  :1B       (make MACC  +1)
130 EC89 C9           LOE257  RET
131                   *
132                   ************************
133                   * TEST A FPT VARIABLE *
134                   ************************
135                   *
136                   * Entry: Variable in MACC.
137                   * Exit:  Z=1: Variable is zero.
138                   *        Z=0: Other flags set on exponent byte
139                   *             of variable.
140                   *        ABCDEHL preserved.
141                   *
142 EC8A C5            FTEST   PUSH  B
143 EC8B D5                    PUSH  D
144 EC8C F5                    PUSH  PSW
145 EC8D E7                    RST   4         Copy MACC to reg A,B,C,D
146 EC8E 15                    DATA  :15
147 EC8F 5F                    MOV   E,A       Exp byte in E
148 EC90 B0                    ORA   B         )
149 EC91 B1                    ORA   C         ) Check if nr is zero
150 EC92 B2                    ORA   D         )
151 EC93 CA98EC                JZ    :EC98     Then quit
152 EC96 7B                    MOV   A,E       Get exp byte
153 EC97 B7                    ORA   A         Set flags on it
154 EC98 D1           FTS10   POP   D
155 EC99 7A                    MOV   A,D
156 EC9A D1                    POP   D
157 EC9B C1                    POP   B
158 EC9C C9                    RET
159                   *
160                   ****************************
161                   * RUN basicfunction SCRN *
162                   ****************************
163                   *
164 EC9D CDF3E5       RSCRN   CALL  :E5F3     Eval given coord
165 ECA0 C5                    PUSH  B
166 ECA1 4F                    MOV   C,A       Y-coord in C
167 ECA2 EF                    RST   5         Ask colour of dot on screen
168 ECA3 27                    DATA  :27       + size graphics screen
169 ECA4 C1                    POP   B
170 ECA5 DA02E6                JC    :E602     Evt run screen error
171 ECA8 C37CEB                JMP   :EB7C     Contents screen loc in MACC
172                   *
173                   *
174                   *  =============
175                   *** LIST HANDLER ***
176                   *  =============
177                   *
178                   * This module lists a program from the textbuffer
179                   * onto the screen (or into other required direction)
180                   *
181                   ************************
182                   * LIST A PROGRAM LINE *
183                   ************************
184                   *
185                   * Entry: BC: Points to start of textline.
186                   * Exit:  BC: Points to start of next line.
187                   *        DEHL preserved, AF corrupted.
```

```
126 EC83 0C                 DATA  :0C
127 EC84 FA89EC             JM    :EC89     Ready if already negative
128 EC87 E7                 RST   4         Else change sign MACC
129 EC88 1B                 DATA  :1B       (make MACC  +1)
130 EC89 C9           LOE257  RET
131                   *
132                   ************************
133                   * TEST A FPT VARIABLE *
134                   ************************
135                   *
136                   * Entry: Variable in MACC.
137                   * Exit:  Z=1: Variable is zero.
138                   *        Z=0: Other flags set on exponent byte
139                   *             of variable.
140                   *        ABCDEHL preserved.
141                   *
142 EC8A C5            FTEST   PUSH  B
143 EC8B D5                    PUSH  D
144 EC8C F5                    PUSH  PSW
145 EC8D E7                    RST   4         Copy MACC to reg A,B,C,D
146 EC8E 15                    DATA  :15
147 EC8F 5F                    MOV   E,A       Exp byte in E
148 EC90 B0                    ORA   B         )
149 EC91 B1                    ORA   C         ) Check if nr is zero
150 EC92 B2                    ORA   D         )
151 EC93 CA98EC                JZ    :EC98     Then quit
152 EC96 7B                    MOV   A,E       Get exp byte
153 EC97 B7                    ORA   A         Set flags on it
154 EC98 D1           FTS10   POP   D
155 EC99 7A                    MOV   A,D
156 EC9A D1                    POP   D
157 EC9B C1                    POP   B
158 EC9C C9                    RET
159                   *
160                   ****************************
161                   * RUN basicfunction SCRN *
162                   ****************************
163                   *
164 EC9D CDF3E5       RSCRN   CALL  :E5F3     Eval given coord
165 ECA0 C5                    PUSH  B
166 ECA1 4F                    MOV   C,A       Y-coord in C
167 ECA2 EF                    RST   5         Ask colour of dot on screen
168 ECA3 27                    DATA  :27       + size graphics screen
169 ECA4 C1                    POP   B
170 ECA5 DA02E6                JC    :E602     Evt run screen error
171 ECA8 C37CEB                JMP   :EB7C     Contents screen loc in MACC
172                   *
173                   *
174                   *  =============
175                   *** LIST HANDLER ***
176                   *  =============
177                   *
178                   * This module lists a program from the textbuffer
179                   * onto the screen (or into other required direction)
180                   *
181                   ************************
182                   * LIST A PROGRAM LINE *
183                   ************************
184                   *
185                   * Entry: BC: Points to start of textline.
186                   * Exit:  BC: Points to start of next line.
187                   *        DEHL preserved, AF corrupted.
```

```
188                     *
189 ECAB D5             SLINE   PUSH  D
190 ECAC E5                     PUSH  H
191 ECAD 03                     INX   B          Pnts to line nr
192 ECAE CDAEEF                 CALL  :EFAE      List line nr
193 ECB1 3E08                   MVI   A,:08
194 ECB3 CD2ADB                 CALL  :DB2A      Cursor to tab 8
195 ECB6 CDCCEC         LOE262  CALL  :ECCC      List statement
196 ECB9 0A                     LDAX  B          Get next byte
197 ECBA B7                     ORA   A
198 ECBB F2C5EC                 JP    :ECC5      If no more statements
199 ECBE CDF5EF                 CALL  :EFF5      Else: print ':'
200 ECC1 3A                     DATA  :3A
201 ECC2 C3B6EC                 JMP   :ECB6      List next statement
202 ECC5 CDF5EF         LOE263  CALL  :EFF5      print car.ret
203 ECC8 0D                     DATA  :0D
204 ECC9 E1                     POP   H
205 ECCA D1                     POP   D
206 ECCB C9                     RET
207                     *
208                     ********************
209                     * LIST A STATEMENT *
210                     ********************
211                     *
212                     * Based on the token in the textbuffer, a particular
213                     * statement will be printed.
214                     * At first, the Basiccommand will be printed. The
215                     * pointers to the particular strings are in a table
216                     * starting at CD8B. The base for the table is CC08;
217                     * the offset is calculated by TOKEN *3.
218                     *
219                     * The databyte after the stringaddress pointer
220                     * indicates which list-routine has to be used for
221                     * the rest of the statement. This byte is a offset
222                     * for table ECF8.
223                     *
224                     * Entry: BC: Points to token.
225                     * Exit:  BC: Points to next statement.
226                     *        AFDEHL corrupted.
227                     *        On stack: Returnaddress from this sub-
228                     *        routine.
229                     *
230 ECCC 0A             SCOM    LDAX  B          Get token
231 ECCD 03                     INX   B          Update pointer
232 ECCE 5F                     MOV   E,A        token in E
233 ECCF 1600                   MVI   D,:00
234 ECD1 2108CC                 LXI   H,:CC08    Startaddr stringtable
235 ECD4 19                     DAD   D          ) Add 3* token
236 ECD5 19                     DAD   D          )
237 ECD6 19                     DAD   D          )
238 ECD7 5E                     MOV   E,M        Get lobyte stringaddr
239 ECD8 23                     INX   H
240 ECD9 56                     MOV   D,M        Get hibyte stringaddr
241 ECDA 23                     INX   H          Point to data after addr
242 ECDB EB                     XCHG             Stringaddr in HL
243 ECDC 7E                     MOV   A,M        Get length byte of string
244 ECDD B7                     ORA   A          Length=0?
245 ECDE CD32DB                 CALL  :DB32      List Basiccmd string
246 ECE1 1A                     LDAX  D          Get data byte after string-
247                                              address
248 ECE2 CAEAEC                 JZ    :ECEA      If length string =0
249 ECE5 FE00                   CPI   :00
```

```
250 ECE7 C46BCE             CNZ   :CE6B     Print space if byte after
251                                         stringaddr <>0
252 ECEA 1A          SCM10  LDAX  D          Get data byte (= offset)
253 ECEB 87                 ADD   A          Offset *2
254 ECEC 5F                 MOV   E,A        in E
255 ECED 1600               MVI   D,:00
256 ECEF 21F8EC             LXI   H,:ECF8    Startaddr table Listroutines
257 ECF2 19                 DAD   D          Add offset
258 ECF3 5E                 MOV   E,M        Get addr in DE
259 ECF4 23                 INX   H
260 ECF5 56                 MOV   D,M
261 ECF6 EB                 XCHG             Addr routine in HL
262 ECF7 E9                 PCHL             Go to this adress
263                 *
264                 ************************************
265                 * POINTERS LIST HANDLING ROUTINES *
266                 ************************************
267                 *
268                 * Table with addresses of listroutines for the
269                 * part of a statement after a token.
270                 *
271                 * Startaddress table is ECF8. The offset (given
272                 * between brackets) is identical to the data
273                 * byte after the addresses in the table on CD8B.
274                 *
275 ECF8 3AED       L0E355 DBL   :ED3A      (00)  nothing more
276 ECFA 41ED              DBL   :ED41      (01)  linenr
277 ECFC 3BED              DBL   :ED3B      (02)  linenr linenr
278                                         .     (not used)
279 ECFE 44ED              DBL   :ED44      (03)  unquoted string
280 ED00 4DED              DBL   :ED4D      (04)  E (E=expr)
281 ED02 47ED              DBL   :ED47      (05)  E,E
282 ED04 56ED              DBL   :ED56      (06)  E E
283 ED06 62ED              DBL   :ED62      (07)  E,E E
284 ED08 5CED              DBL   :ED5C      (08)  E,E E,E E
285 ED0A 50ED              DBL   :ED50      (09)  E E E E
286 ED0C 68ED              DBL   :ED68      (0A)  E
287 ED0E 7AED              DBL   :ED7A      (0B)  liner-linenr
288 ED10 84ED              DBL   :ED84      (0C)  sound
289 ED12 9BED              DBL   :ED9B      (0D)  noise
290 ED14 A4ED              DBL   :EDA4      (0E)  envelope
291 ED16 C1ED              DBL   :EDC1      (0F)  mode
292 ED18 D9ED              DBL   :EDD9      (10)  input <string>
293 ED1A E0ED              DBL   :EDE0      (11)  input/read/dim
294 ED1C F0ED              DBL   :EDF0      (12)  (not used [*])
295 ED1E FFED              DBL   :EDFF      (13)  let
296 ED20 09EE              DBL   :EE09      (14)  if then <E>
297 ED22 1AEE              DBL   :EE1A      (15)  if goto <linenr>
298 ED24 25EE              DBL   :EE25      (16)  if then <linenr>
299 ED26 30EE              DBL   :EE30      (17)  for to step
300 ED28 4FEE              DBL   :EE4F      (18)  next
301 ED2A 52EE              DBL   :EE52      (19)  print
302 ED2C 66EE              DBL   :EE66      (1A)  on goto
303 ED2E 71EE              DBL   :EE71      (1B)  on gosub
304 ED30 87EE              DBL   :EE87      (1C)  callm
305 ED32 94EE              DBL   :EE94      (1D)  (not used [*])
306 ED34 9ED8              DBL   :D89E      (1E)  savea/loada
307                 *
308                 * The vectors marked with [*] are no pointers to
309                 * LIST routines.
310                 *
                            DATA  :FF
```

```
312 ED37 FF                DATA  :FF
313 ED38 FF                DATA  :FF
314 ED39 FF                DATA  :FF
315                        *
316                        ********************************
317                        * LIST NO FURTHER EXPRESSIONS *
318                        ********************************
319                        *
320 ED3A C9                SCN1    RET
321                        *
322                        ****************************
323                        * LIST 1 OR 2 LINENUMBERS *
324                        ****************************
325                        *
326                        * SCN2: List 1 line number.
327                        * SCN3: List 2 line numbers, separated by space.
328                        *       (This last entry is not used).
329                        *
330                        * Exit: BC updated, DE preserved, AFHL corrupted.
331                        *
332 ED3B CDAEEF      SCN3    CALL  :EFAE     List linenr
333 ED3E CD6BCE              CALL  :CE6B     Print space
334 ED41 C3AEEF      SCN2    JMP   :EFAE     List linenr
335                        *
336                        ************************
337                        * LIST UNQUOTED STRING *
338                        ************************
339                        *
340 ED44 C3EDEF      SCN5    JMP   :EFED     List unquoted string
341                        *
342                        ************************
343                        * LIST <EXPR>,<EXPR> *
344                        ************************
345                        *
346                        * Exit: BC updated, DE preserved, AFHL corrupted.
347                        *
348 ED47 CDA2EE      SCN7    CALL  :EEA2     List <expr>
349 ED4A CD70CE      SC0EX   CALL  :CE70     Print ','
350 ED4D C3A2EE      SCN6    JMP   :EEA2     List <expr>
351                        *
352                        ************************************
353                        * LIST <EXPR> <EXPR> <EXPR> <EXPR> *
354                        ************************************
355                        *
356                        * Exit: BC updated, DE preserved, AFHL corrupted.
357                        *
358 ED50 CDFCEF      SCN11   CALL  :EFFC     List <expr>; print space
359 ED53 CDFCEF      S3EXP   CALL  :EFFC     Idem
360 ED56 CDFCEF      SCN8    CALL  :EFFC     Idem
361 ED59 C3A2EE              JMP   :EEA2     List <expr>
362                        *
363                        ********************************************
364                        * LIST <EXPR>,<EXPR> <EXPR>,<EXPR> <EXPR> *
365                        ********************************************
366                        *
367                        * Exit: BC updated, DE preserved, AFHL corrupted.
368                        *
369 ED5C CD47ED      SCN10   CALL  :ED47     List <expr>,<expr>
370 ED5F CD6BCE              CALL  :CE6B     Print space
371 ED62 CD47ED      SCN9    CALL  :ED47     List <expr>,<expr>
372 ED65 CD6BCE      SCSEX   CALL  :CE6B     Print space
373 ED68 C3A2EE      LOE343  JMP   :EEA2     List <expr>
```

```
374                     *
375                     ********************************
376                     * LIST <EXPR>,<EXPR>(,<EXPR>) *
377                     ********************************
378                     *
379 ED6B CD47ED      SCN12   CALL  :ED47     List <expr>,<expr>
380 ED6E 0A          S1210   LDAX  B         Get next byte
381 ED6F FEFF              CPI   :FF         Terminator?
382 ED71 03                INX   B
383 ED72 C8                RZ              Then abort
384 ED73 0B                DCX   B
385 ED74 CD70CE            CALL  :CE70     Print ','
386 ED77 C3A2EE            JMP   :EEA2     List <expr>
387                     *
388                     ****************************
389                     * LIST <LINENR>-<LINENR> *
390                     ****************************
391                     *
392                     * Exit: BC updated, DE preserved, AFHL corrupted.
393                     *
394 ED7A CDAEEF      SCN13   CALL  :EFAE     List linenr
395 ED7D CDF5EF            CALL  :EFF5     Print '-'
396 ED80 2D                DATA  :2D
397 ED81 C3AEEF            JMP   :EFAE     List linenr
398                     *
399                     ***********************************
400                     * LIST EXPRESSION AFTER 'SOUND' *
401                     ***********************************
402                     *
403                     * Exit: BC updated, AFHL corrupted.
404                     *       DE: preserved if ON, corrupted if OFF.
405                     *
406 ED84 0A          SCN14   LDAX  B         Get byte
407 ED85 FEFF              CPI   :FF         OFF sign?
408 ED87 C4FCEF            CNZ   :EFFC     If not: List <expr>, print
409                                         space
410 ED8A 0A                LDAX  B         Get next byte
411 ED8B FEFF              CPI   :FF         OFF sign?
412 ED8D C250ED            JNZ   :ED50     If not: List <expr> <expr>
413                                         <expr> <expr>; abort
414 ED90 CD78CE      S1410   CALL  :CE78     Else: print 'OFF'
415 ED93 97ED              DBL   :ED97
416 ED95 03                INX   B
417 ED96 C9                RET
418                     *
419                     * DATA:
420                     *
421 ED97 03          LOE354  DATA  :03
422 ED98 4F                DATA  :4F         O
423 ED99 46                DATA  :46         F
424 ED9A 46                DATA  :46         F
425                     *
426                     ***********************************
427                     * LIST EXPRESSION AFTER 'NOISE' *
428                     ***********************************
429                     *
430                     * Exit: BC updated, E preserved, AFDHL corrupted.
431                     *
432 ED9B 0A          SC14A   LDAX  B         Get 1st byte NCB
433 ED9C FEFF              CPI   :FF         OFF sign?
434 ED9E CA90ED            JZ    :ED90     Then print 'OFF', abort
435 EDA1 C356ED            JMP   :ED56     Else: List <expr> <expr>
```

```
436                     *
437                     ************************************
438                     * LIST EXPRESSION AFTER 'ENVELOPE' *
439                     ************************************
440                     *
441                     * Exit: BC updated, E preserved, AFDHL corrupted.
442                     *
443 EDA4 CDFCEF      SCN15   CALL  :EFFC     List <expr>, print
444                                           space
445 EDA7 0A                 LDAX  B         Get length of expr
446 EDA8 03                 INX   B
447 EDA9 57                 MOV   D,A       into D
448 EDAA 15          S1510   DCR   D
449 EDAB FAB8ED             JM    :EDB8     If ready
450 EDAE CD47ED             CALL  :ED47     List <V>,<T>
451 EDB1 CDF5EF             CALL  :EFF5     Print ';'
452 EDB4 3B                 DATA  :3B
453 EDB5 C3AAED             JMP   :EDAA     Next <V>,<T>
454 EDB8 0A          S1520   LDAX  B         Get last byte of expr
455 EDB9 03                 INX   B
456 EDBA FEFF               CPI   :FF       Terminator?
457 EDBC C8                 RZ              Then abort
458 EDBD 0B                 DCX   B
459 EDBE C3A2EE             JMP   :EEA2     List <expr>
460                     *
461                     ********************************
462                     * LIST EXPRESSION AFTER 'MODE' *
463                     ********************************
464                     *
465 EDC1 0A          SCN16   LDAX  B         Get mode byte
466 EDC2 03                 INX   B
467 EDC3 1630               MVI   D,:30
468 EDC5 B7                 ORA   A         Mode 0 (FF) ?
469 EDC6 FAD4ED             JM    :EDD4     Then print '0'
470 EDC9 1F                 RAR             CY=1 if A-mode
471 EDCA 3C          RSA10   INR   A
472 EDCB F5                 PUSH  PSW
473 EDCC 82                 ADD   D         Convert to ASCII
474 EDCD CD60DD             CALL  :DD60     Print modenr
475 EDD0 F1                 POP   PSW
476 EDD1 3F                 CMC
477 EDD2 1641               MVI   D,:41     Prepare print 'A'
478 EDD4 7A          S1610   MOV   A,D
479 EDD5 D460DD             CNC   :DD60     Print 'A' if A-mode
480 EDD8 C9                 RET
481                     *
482                     ************************************************
483                     * LIST EXPRESSION AFTER 'INPUT'-'READ'-'DIM' *
484                     ************************************************
485                     *
486                     * Input with string:
487
488 EDD9 CDA2EE      SCN17   CALL  :EEA2     List string
489 EDDC CDF5EF             CALL  :EFF5     Print ';'
490 EDDF 3B                 DATA  :3B
491
492                     * Rest:
493
494 EDE0 0A          SCN18   LDAX  B         Get nr of variables
495 EDE1 03                 INX   B
496 EDE2 57                 MOV   D,A       into D
497 EDE3 D5          S1810   PUSH  D
```

```
498 EDE4 CDFCEE             CALL  :EEFC     List variable reference
499 EDE7 D1                 POP   D
500 EDE8 15                 DCR   D         Decr nr of variables
501 EDE9 C8                 RZ              Abort if ready
502 EDEA CD70CE             CALL  :CE70     Print ','
503 EDED C3E3ED             JMP   :EDE3     List next variable
504                 *
505                 ****************************************
506                 * part of RUN 'DIM': CALC. REQD. SPACE *
507                 ****************************************
508                 *
509 EDF0 C28FDE    RDM40   JNZ   :DE8F     If length element <= 254:
510                                         then HL=HL*A
511 EDF3 7C                 MOV   A,H       Else:
512 EDF4 B7                 ORA   A         Set flags on hibyte
513 EDF5 65                 MOV   H,L
514 EDF6 2E00               MVI   L,:00
515 EDF8 C8                 RZ              Abort if H=0
516 EDF9 37                 STC             Else: CY=1, L into H
517 EDFA C9                 RET
518                 *
519                 ************************
520                 * RUBBISH - (not used) *
521                 ************************
522                 *
523 EDFB CE          L0E352  DATA  :CE
524 EDFC C3                 DATA  :C3
525 EDFD F4                 DATA  :F4
526 EDFE ED                 DATA  :ED
527                 *
528                 ********************************
529                 * LIST EXPRESSION AFTER 'LET' *
530                 ********************************
531                 *
532                 * Entry: BC: Points to assign statement.
533                 * Exit:  BC updated, AFDEHL corrupted.
534                 *
535 EDFF CDFCEE    SCN20   CALL  :EEFC     List lefthand variable
536                                         reference
537 EE02 CDF5EF             CALL  :EFF5     Print '='
538 EE05 3D                 DATA  :3D
539 EE06 C3A2EE             JMP   :EEA2     List righthand expr
540                 *
541 EE09                    END
```

```
****************************
* S Y M B O L   T A B L E *
****************************
```

FTEST	EC8A	FTS10	EC98	L0E252	EC23	L0E257	EC89
L0E262	EC86	L0E263	ECC5	L0E343	ED68	L0E352	EDFB
L0E354	ED97	L0E355	ECF8	MPT46	EC6D	RDM40	EDF0
RPEEK	EC16	RPI	EC1D	RRD10	EC44	RRND	EC27
RSA10	EDCA	RSCRN	EC9D	RSGN	EC7B	S1210	ED6E
S1410	ED90	S1510	EDAA	S1520	EDB8	S1610	EDD4
S1810	EDE3	S3EXP	ED53	SC14A	ED9B	SCM10	ECEA
SCN1	ED3A	SCN10	ED5C	SCN11	ED50	SCN12	ED6B
SCN13	ED7A	SCN14	ED84	SCN15	EDA4	SCN16	EDC1
SCN17	EDD9	SCN18	EDE0	SCN2	ED41	SCN20	EDFF
SCN3	ED3B	SCN5	ED44	SCN6	ED4D	SCN7	ED47
SCN8	ED56	SCN9	ED62	SCOEX	ED4A	SCOM	ECCC

```
002                             ORG   :EE09
003                     *
004                     *
005                     *
006                     ************************************
007                     * LIST EXPRESSION AFTER 'IF' (1) *
008                     ************************************
009                     *
010                     * Lists '<expr> THEN <expr>'.
011                     *
012 EE09 CDFCEF         SCN21   CALL  :EFFC     List <expr>; print space
013 EE0C CD78CE                 CALL  :CE78     Print 'THEN'
014 EE0F 15EE           MPT17   DBL   :EE15
015 EE11 03                     INX   B
016 EE12 C3CCEC                 JMP   :ECCC     List statement
017                     *
018                     * DATA :
019                     *
020 EE15 04             L0E350  DATA  :04
021 EE16 54                     DATA  :54       T
022 EE17 48                     DATA  :48       H
023 EE18 45                     DATA  :45       E
024 EE19 4E                     DATA  :4E       N
025                     *
026                     ************************************
027                     * LIST EXPRESSION AFTER 'IF' (2) *
028                     ************************************
029                     *
030                     * Lists '<expr> GOTO <linenr>'.
031                     *
032 EE1A CDFCEF         SCN22   CALL  :EFFC     List <expr>; print space
033 EE1D CD78CE                 CALL  :CE78     Print 'GOTO'
034 EE20 F9CB                   DBL   :CBF9
035 EE22 C3AEEF                 JMP   :EFAE     List linenr
036                     *
037                     ************************************
038                     * LIST EXPRESSION AFTER 'IF' (3) *
039                     ************************************
040                     *
041                     * Lists '<expr> THEN <linenr>'.
042                     *
043 EE25 CDFCEF         SC22A   CALL  :EFFC     List <expr>; print space
044 EE28 CD78CE                 CALL  :CE78     Print 'THEN'
045 EE2B 15EE                   DBL   :EE15
046 EE2D C3AEEF                 JMP   :EFAE     List linenr
047                     *
048                     ********************************
049                     * LIST EXPRESSION AFTER 'FOR' *
050                     ********************************
051                     *
052                     * Lists '<LET statement> TO <expr> (STEP <expr>)'.
053                     *
054 EE30 CDFFED         SCN23   CALL  :EDFF     List expr of LET statement
055 EE33 CD6BCE                 CALL  :CE6B     Print space
056 EE36 CD78CE                 CALL  :CE78     Print 'TO'
057 EE39 4CEE                   DBL   :EE4C
058 EE3B CDA2EE                 CALL  :EEA2     List <expr>
059 EE3E 0A                     LDAX  B         Get next byte
060 EE3F 03                     INX   B
061 EE40 FEFF                   CPI   :FF       Terminator?
062 EE42 C8                     RZ              Then abort
063 EE43 0B                     DCX   B         Else:
```

```
064 EE44 CD75CE             CALL  :CE75     Print 'STEP'
065 EE47 3DCD               DBL   :CD3D
066 EE49 C3A2EE             JMP   :EEA2     List <expr>
067                 *
068                 * DATA:
069                 *
070 EE4C 02         RLA20   DATA  :02
071 EE4D 54                 DATA  :54       T
072 EE4E 4F                 DATA  :4F       O
073                 *
074                 ************************************
075                 * LIST EXPRESSION AFTER 'NEXT' *
076                 ************************************
077                 *
078 EE4F C3FCEE     SCN24   JMP   :EEFC     List var.ref.
079                 *
080                 ************************************
081                 * LIST EXPRESSIONS AFTER 'PRINT' *
082                 ************************************
083                 *
084 EE52 0A         SCN25   LDAX  B         Get nr of expr
085 EE53 03                 INX   B
086 EE54 57                 MOV   D,A       into D
087 EE55 15         S2510   DCR   D
088 EE56 F8                 RM              Abort if ready
089 EE57 03                 INX   B
090 EE58 CDA2EE             CALL  :EEA2     List <expr>
091 EE5B 0A                 LDAX  B         Get next byte
092 EE5C 03                 INX   B
093 EE5D FEFF               CPI   :FF       Terminator?
094 EE5F C8                 RZ              Then abort
095 EE60 CD60DD             CALL  :DD60     Else: print this byte
096 EE63 C355EE             JMP   :EE55     List next expr
097                 *
098                 ************************************
099                 * LIST EXPRESSION AFTER 'ON' (1) *
100                 ************************************
101                 *
102                 * Lists '<expr> GOTO <linenrs>'.
103                 *
104 EE66 CDFCEF     SCN26   CALL  :EFFC     List <expr>; print space
105 EE69 CD78CE             CALL  :CE78     Print 'GOTO'
106 EE6C F9CB               DBL   :CBF9
107 EE6E C379EE             JMP   :EE79     List linenrs
108                 *
109                 ************************************
110                 * LIST EXPRESSION AFTER 'ON' (2) *
111                 ************************************
112                 *
113                 * Lists '<expr> GOSUB <linenrs>'.
114                 *
115 EE71 CDFCEF     SCN27   CALL  :EFFC     List <expr>; print space
116 EE74 CD78CE             CALL  :CE78     Print 'GOSUB'
117 EE77 01CC               DBL   :CC01
118 EE79 0A         S2710   LDAX  B         Get nr of linenrs
119 EE7A 03                 INX   B
120 EE7B 57                 MOV   D,A       into D
121 EE7C CDAEEF     S2720   CALL  :EFAE     List linenr
122 EE7F 15                 DCR   D
123 EE80 C8                 RZ              Abort if ready
124 EE81 CD70CE             CALL  :CE70     Print ','
125 EE84 C37CEE             JMP   :EE7C     List next linenr
```

```
                        *
                        ***********************************
                        * LIST EXPRESSION AFTER 'CALLM' *
                        ***********************************
                        *
EE87 CDA2EE             SCN28   CALL  :EEA2     List <expr>
EE8A C36EED                     JMP   :ED6E     Print ','; List next expr
                        *
                        **********************
                        * SET I/O DIRECTION *
                        **********************
                        *
                        * Part of RESET (C719). Only used for A = 0.
                        * Depending on A, the input switch #0296 and
                        * the output switch #0131 are set.
                        * Default DINC is RS232.
                        *
                        *             INSW:       OTSW:
                        *       A=0: keyboard     screen/RS232
                        *       A=1: DINC         screen
                        *       A=2: DINC         editbuffer
                        *       A=3: DINC         DOUTC
                        *
EE8D 329602             MPT02   STA   :0296     Select keyb or DINC
EE90 323101                     STA   :0131     Select screen/RS232/edit/
                                                DOUTC
EE93 C9                         RET
                        *
                        ***************
                        * SET VOLUMES *
                        ***************
                        *
                        * Part of RUN TALK (CD64).
                        *
                        * Entry: A:  Parameter code.
                        *        HL: Pointer to volume byte.
                        *
EE94 E5                 RTK40   PUSH  H         Save pntr to volume
EE95 6E                         MOV   L,M       Volume in L
EE96 260F                       MVI   H,:0F
EE98 1F                         RAR             Check parameter code
EE99 D27CE6                     JNC   :E67C     Jump if channel 0/2

                        * If channel 1/N:

EE9C 29                         DAD   H
EE9D 29                         DAD   H
EE9E 29                         DAD   H
EE9F C37BE6                     JMP   :E67B     Continu
                        *
                        ************************
                        * LIST AN EXPRESSION *
                        ************************
                        *
                        * Entry: BC:   Points to expression in program.
                        *        (BC): 1.... Expr starts with operator.
                        *              01... Variable reference.
                        *              001.. Function call.
                        *              Else  Constant.
                        * Exit:  BC points after expression.
                        *        DE preserved, AFHL corrupted.
                        *
```

```
188 EEA2 D5          SCEXP   PUSH  D
189 EEA3 0A                  LDAX  B          Get opcode
190 EEA4 B7                  ORA   A
191 EEA5 F2DFEE              JP    :EEDF      If no starting operator
192
193                  * If starting with operator:
194
195 EEA8 03                  INX   B
196 EEA9 E61F                ANI   :1F        Only 5 bits of opcode
197 EEAB FE1A                CPI   :1A        '(' ?
198 EEAD F5                  PUSH  PSW        Save opcode
199 EEAE DCA2EE              CC    :EEA2      List expr if binary
200                                           operation
201 EEB1 2186CF              LXI   H,:CF86    Addr table opcode strings
202 EEB4 54          LOE300  MOV   D,H        ) in DE
203 EEB5 5D                  MOV   E,L        )
204 EEB6 CD39DE              CALL  :DE39      HL points after table
205 EEB9 F1                  POP   PSW        Get opcode
206 EEBA F5                  PUSH  PSW
207 EEBB AE                  XRA   M          Comp it with table
208 EEBC 23                  INX   H
209 EEBD E61F                ANI   :1F
210 EEBF C2B4EE              JNZ   :EEB4      Check next opcode if not
211                                           found
212 EEC2 EB                  XCHG             If found: addr string in HL
213 EEC3 23                  INX   H
214 EEC4 7E                  MOV   A,M        Get 1st char
215 EEC5 2B                  DCX   H          Pnts to length
216 EEC6 CD02DE              CALL  :DE02      Check if upper case char
217 EEC9 F5                  PUSH  PSW
218 EECA DC6BCE              CC    :CE6B      Print space if 1st char is
219                                           a letter
220 EECD CD32DB              CALL  :DB32      Print string from table
221 EED0 F1                  POP   PSW
222 EED1 DC6BCE              CC    :CE6B      Print space if 1st char was
223                                           a letter
224 EED4 CDA2EE              CALL  :EEA2      List remaining operand
225 EED7 F1                  POP   PSW        Get orig. opcode
226 EED8 FE1A                CPI   :1A        Was it '(' ?
227 EEDA CC55EF              CZ    :EF55      Then print ')'
228 EEDD D1                  POP   D
229 EEDE C9                  RET
230
231                  * Not starting with operator:
232
233 EEDF 07          LOE301  RLC
234 EEE0 07                  RLC
235 EEE1 DAF2EE              JC    :EEF2      Jump if var.ref
236 EEE4 07                  RLC
237 EEE5 DAEDEE              JC    :EEED      Jump if function call
238
239                  * If constant:
240
241 EEE8 CD84EF               CALL  :EF84     List constant
242 EEEB D1                  POP   D
243 EEEC C9                  RET
244
245                  * If function call:
246
247 EEED CD5AEF       LOE302  CALL  :EF5A     List function reference
248 EEF0 D1                  POP   D
249 EEF1 C9                  RET
```

```
250
251                     * If variable reference:
252
253 EEF2 CDFCEE      LOE303  CALL   :EEFC      List a var.reference
254                                            (array with arguments)
255 EEF5 D1                  POP    D
256 EEF6 C9                  RET
257                     *
258                     ******************************
259                     * LIST A VARIABLE REFERENCE *
260                     ******************************
261                     *
262                     * SCARN:  Entry for arrays without arguments (name
263                     *         only).
264                     * LOE305: Entry for arrays with arguments.
265                     *
266                     * Entry: BC points to variable reference in program.
267                     *
268 EEF7 16BF          SCARN   MVI    D,:BF      Set mask 'no arg'
269 EEF9 C3FEEE              JMP    :EEFE
270                     *
271 EEFC 16FF          LOE305  MVI    D,:FF      Set mask 'with arg'
272 EEFE D5              LOE306  PUSH   D          Save mask
273 EEFF 0A                    LDAX   B          Get byte
274 EF00 03                    INX    B
275 EF01 E63F                  ANI    :3F        Skip bit 6,7
276 EF03 57                    MOV    D,A        Rest in D
277 EF04 0A                    LDAX   B          Get next byte
278 EF05 03                    INX    B
279 EF06 5F                    MOV    E,A        in E (Now DE is offset
280                                               of start of symtab)
281 EF07 2AA102                LHLD   :02A1      Get start symtab
282 EF0A 19                    DAD    D          Add offset from start
283 EF0B D1                    POP    D          Get mask
284 EF0C E5                    PUSH   H          Save var.addr in symtab
285 EF0D CD95CA                CALL   :CA95      Find name in symtab
286 EF10 7E                    MOV    A,M        Get T/L byte
287 EF11 A2                    ANA    D          AND with mask
288 EF12 F5                    PUSH   PSW
289 EF13 23                    INX    H
290 EF14 E60F                  ANI    :0F        Get length
291 EF16 5E                    MOV    E,M        Get 1st byte of name in E
292 EF17 CD44DB                CALL   :DB44      List name; addr in HL,
293                                               length in A
294 EF1A 1600                  MVI    D,:00
295 EF1C 213402                LXI    H,:0234    Startaddr for IMPTAB
296 EF1F 19                    DAD    D          Addr var.type in IMPTAB
297 EF20 F1                    POP    PSW        Get mask
298 EF21 F5                    PUSH   PSW
299 EF22 E630                  ANI    :30        Bits 4,5 only
300 EF24 BE                    CMP    M          Comp with IMPTAB
301 EF25 CA3CEF                JZ     :EF3C      Jump if identical
302 EF28 FE00                  CPI    :00        FPT ?
303 EF2A 1621                  MVI    D,:21      Then D: '!'
304 EF2C CA38EF                JZ     :EF38
305 EF2F FE10                  CPI    :10        INT ?
306 EF31 1625                  MVI    D,:25      Then D: '%'
307 EF33 CA38EF                JZ     :EF38
308 EF36 1624                  MVI    D,:24      Else: D: '$'
309 EF38 7A          LOE307  MOV    A,D
310 EF39 CD60DD                CALL   :DD60      Print type sign
311 EF3C F1          LOE308  POP    PSW        Get mask
```

```
312 EF3D E640                ANI   :40       Bit 6 only
313 EF3F E1                  POP   H         Get addr of string
314 EF40 C8                  RZ
315
316                  * Bit 6=1 (array with arguments):
317
318 EF41 0A                   LDAX  B         Get nr of expressions
319 EF42 03                   INX   B
320 EF43 57                   MOV   D,A       in D
321 EF44 CDF5EF               CALL  :EFF5     Print '('
322 EF47 28                   DATA  :28
323 EF48 03          L0E309   INX   B
324 EF49 D5                   PUSH  D
325 EF4A CDA2EE               CALL  :EEA2     List expression
326 EF4D D1                   POP   D
327 EF4E 15                   DCR   D         Ready ?
328 EF4F C470CE               CNZ   :CE70     If not: print ','
329 EF52 C248EF               JNZ   :EF48     and list next expr
330 EF55 CDF5EF      L0E310   CALL  :EFF5     If ready: Print ')'
331 EF58 29                   DATA  :29
332 EF59 C9                   RET
333                  *
334                  ********************************
335                  * LIST A FUNCTION REFERENCE *
336                  ********************************
337                  *
338                  * Finds functionname in table with startaddress
339                  * #CFE6 and prints it. Eventual arguments are
340                  * printed between brackets.
341                  *
342                  * Entry: BC points to function code (#20).
343                  * Exit:  BC updated, AFEHL preserved, D=0.
344                  *
345 EF5A 03          SFUN     INX   B
346 EF5B 0A                   LDAX  B
347 EF5C 03                   INX   B
348 EF5D 57                   MOV   D,A
349 EF5E 21E6CF               LXI   H,:CFE6   Startaddr function table
350 EF61 15          L0E312   DCR   D
351 EF62 FA6BEF               JM    :EF6B     If found
352 EF65 CDAECA               CALL  :CAAE     Calc addr next string in tab
353 EF68 C361EF               JMP   :EF61     Test next function name
354 EF6B CD32DB      L0E313   CALL  :DB32     List function name
355 EF6E 7E                   MOV   A,M       Get byte after string
356 EF6F E60F                 ANI   :0F       Only nr of following args
357 EF71 57                   MOV   D,A       in D
358 EF72 C8                   RZ              Abort if no arguments
359 EF73 CDF5EF               CALL  :EFF5     Print '('
360 EF76 28                   DATA  :28
361 EF77 CDA2EE      L0E314   CALL  :EEA2     List expression
362 EF7A 15                   DCR   D         Decr nr of arg
363 EF7B C470CE               CNZ   :CE70     If <>0, print ','
364 EF7E C277EF               JNZ   :EF77     and list next expr
365 EF81 C355EF               JMP   :EF55     If ready: print ')'
366                  *
367                  ********************
368                  * LIST A CONSTANT *
369                  ********************
370                  *
371                  * The constant is decoded to ASCII, prettied and
372                  * printed.
373                  *
```

```
                    * Codes:   #10: FPT       #18: Quoted string
                    *          #14: INT       #19: Unquoted string
                    *          #15: HEX
                    *
                    * Entry: BC: Points to constant in program.
                    * Exit:  BC updated, DE preserved, AF corrupted.
                    *        HL: points after end of printed string.
                    *
EF84 0A             SCON    LDAX  B         Get type of constant
EF85 03                     INX   B
EF86 60                     MOV   H,B       ) Addr constant in HL
EF87 69                     MOV   L,C       )

                    * If string:

EF88 FE18                   CPI   :18       Quoted string ?
EF8A CAE1EF                 JZ    :EFE1     Then list it
EF8D FE19                   CPI   :19       Unquoted string ?
EF8F CAEDEF                 JZ    :EFED     Then list it

                    * If number:

EF92 E7                     RST   4         Copy constant value to MACC
EF93 0C                     DATA  :0C
EF94 FE10                   CPI   :10       FPT ?
EF96 CAA8EF                 JZ    :EFA8     Then list FPT value
EF99 FE14                   CPI   :14       INT ?
EF9B F5                     PUSH  PSW
EF9C CCBDEF                 CZ    :EFBD     List INT value
EF9F F1                     POP   PSW
EFA0 C4DAEF                 CNZ   :EFDA     Else: list HEX value
EFA3 03             SC010   INX   B
EFA4 03                     INX   B
EFA5 03                     INX   B
EFA6 03                     INX   B         BC points after constant
EFA7 C9                     RET
EFA8 CDD4EF         SC020   CALL  :EFD4     List FPT value
EFAB C3A3EF                 JMP   :EFA3
                    *
                    **********************
                    * LIST A LINENUMBER *
                    **********************
                    *
                    * Entry: BC: Points to linenumber.
                    * Exit:  BC updated, DE preserved, AFHL corrupted.
                    *
EFAE 0A             LOE318  LDAX  B         Get hibyte linenr
EFAF 03                     INX   B
EFB0 67                     MOV   H,A       in H
EFB1 0A                     LDAX  B         Get lobyte linenr
EFB2 03                     INX   B
EFB3 6F                     MOV   L,A       in L
EFB4 CD46EB         SLN10   CALL  :EB46     Linenr into MACC
EFB7 7C                     MOV   A,H
EFB8 B5                     ORA   L         Linenr <>0 ?
EFB9 C4BDEF                 CNZ   :EFBD     Then list linenr
EFBC C9                     RET
                    *
                    ****************************************
                    * LIST A INT VALUE OF MACC CONTENTS *
                    ****************************************
                    *
```

```
                    * The value is the contents of the MACC, prepared
                    * for output, and moved into the outputbuffer
                    * DECBUF (#00E3). A Leading space is omitted.
                    *
                    * Exit: BCDE preserved. A corrupted.
                    *       HL: Points after string in DECBUF.
                    *
EFBD CD5FDB         SCINT   CALL  :DB5F     Convert MACC for INT
                                            output into DECBUF
EFC0 2A33C0         SSSPC   LHLD  :C033     Get addr DECBUF
EFC3 D5                     PUSH  D
EFC4 56                     MOV   D,M       String length in D
EFC5 23                     INX   H
EFC6 7E                     MOV   A,M       1st char in A
EFC7 FE20                   CPI   :20       Space ?
EFC9 C2CEEF                 JNZ   :EFCE     Jump if no leading space
EFCC 23                     INX   H         ) Omit leading space
EFCD 15                     DCR   D         )
EFCE 7A             SSS10   MOV   A,D       Get nr of char in A
EFCF D1                     POP   D
EFD0 CD44DB                 CALL  :DB44     Print contents DECBUF
EFD3 C9                     RET
                    *
                    ************************************
                    * LIST FPT VALUE OF CONTENTS MACC *
                    ************************************
                    *
                    * Exit: BCDE preserved. AF corrupted.
                    *       HL points after string in DECBUF.
                    *
EFD4 CD9BCE         SCFPT   CALL  :CE9B     Convert MACC for FPT output
EFD7 C3C0EF                 JMP   :EFC0     List its contents
                    *
                    ************************************
                    * LIST HEX VALUE OF CONTENTS MACC *
                    ************************************
                    *
                    * Exit: BCDE preserved. AF corrupted.
                    *       HL points after string in DECBUF.
                    *
EFDA CDF5EF         SCHEX   CALL  :EFF5     Print '#'
EFDD 23                     DATA  :23
EFDE C34ADB                 JMP   :DB4A     List in hex
                    *
                    *************************
                    * LIST A QUOTED STRING *
                    *************************
                    *
                    * Entry: BC: Points to string.
                    * Exit:  BC and HL point after string.
                    *        AF corrupted. DE preserved.
                    *
EFE1 CDF5EF         SQTS    CALL  :EFF5     Print "
EFE4 22                     DATA  :22
EFE5 CDEDEF                 CALL  :EFED     List string
EFE8 CDF5EF                 CALL  :EFF5     Print "
EFEB 22                     DATA  :22
EFEC C9                     RET
                    *
                    *************************
                    * LIST UNQUOTED STRING *
                    *************************
```

```
498                     *
499                     * Entry: BC points to string.
500                     * Exit:  BC annd HL point after string.
501                     *        AFDE preserved.
502                     *
503 EFED 60             SUQTS   MOV   H,B       ) Stringaddr in HL
504 EFEE 69                     MOV   L,C       )
505 EFEF CD32DB                 CALL  :DB32     List string
506 EFF2 44                     MOV   B,H       )
507 EFF3 4D                     MOV   C,L       ) Addr after string in BC
508 EFF4 C9                     RET
509                     *
510                     ********************
511                     * LIST CHARACTER *
512                     ********************
513                     *
514                     * Entry: On stack: The address of the character to
515                     *                  be printed.
516                     * Exit:  BCDEHL preserved. F corrupted.
517                     *        A: Character printed.
518                     *
519 EFF5 E3             SCHRI   XTHL            Get addr from stack
520 EFF6 7E                     MOV   A,M       Get char
521 EFF7 23                     INX   H         HL pnts after char
522 EFF8 E3                     XTHL            Returnaddr on stack
523 EFF9 C360DD                 JMP   :DD60     List char.
524                     *
525                     ***********************************
526                     * LIST EXPRESSION; PRINT SPACE *
527                     ***********************************
528                     *
529 EFFC C368CE         SEXPS   JMP   :CE68     List expr, print space
530                     *
531 EFFF 3E                     DATA  :3E
532                     *
533                     *
534                     *
535 F000                       END
```

```
****************************
* S Y M B O L   T A B L E *
****************************

L0E300 EEB4   L0E301 EEDF   L0E302 EEED   L0E303 EEF2
L0E305 EEFC   L0E306 EEFE   L0E307 EF38   L0E308 EF3C
L0E309 EF48   L0E310 EF55   L0E312 EF61   L0E313 EF6B
L0E314 EF77   L0E318 EFAE   L0E350 EE15   MPT02  EE8D
MPT17  EE0F   RLA20  EE4C   RTK40  EE94   S2510  EE55
S2710  EE79   S2720  EE7C   SC010  EFA3   SC020  EFA8
SC22A  EE25   SCARN  EEF7   SCEXP  EEA2   SCFPT  EFD4
SCHEX  EFDA   SCHRI  EFF5   SCINT  EFBD   SCN21  EE09
SCN22  EE1A   SCN23  EE30   SCN24  EE4F   SCN25  EE52
SCN26  EE66   SCN27  EE71   SCN28  EE87   SCON   EF84
SEXPS  EFFC   SFUN   EF5A   SLN10  EFB4   SQTS   EFE1
SSS10  EFCE   SSSPC  EFC0   SUQTS  EFED
```

```
002                         ORG   :E000
003                 *
004                 *
005                 *
006                 *   ==================
007                 *** MATH./SOUND PACKAGE ***
008                 *   ==================
009                 *
010                 * The sound package starts at 1EE6E.
011                 *
012                 *   =============
013                 *** MATH. PACKAGE ***
014                 *   =============
015                 *
016                 * Called by RST 4; DATA XX. XX indicates the
017                 * offset from #E000 for the different entrypoints.
018                 *
019                 * The routines jumped to by RST 4; DATA 7B-F3 are
020                 * identical to RST 4; DATA 00-78, but for use with
021                 * the AMD9511A Math.chip.
022                 * Which routines are used, depends on the offset
023                 * in RAM address #00D4.
024                 *
025                 * The accumulator is the MACC (00D5-00D8) or the
026                 * math.chip accu MTOS.
027                 *
028                 ************************
029                 * SOFTWARE ENTRYPOINTS *
030                 ************************
031                 *
032                 * #00D4 contains offset 00.
033                 *
034                 SVECA
035 E000 C3AAED    MFADD   JMP   :EDAA     FPT addition
036 E003 C3B4ED    MFSUB   JMP   :EDB4     FPT subtraction
037 E006 C3FEE0    MFMUL   JMP   :E0FE     FPT multiplication
038 E009 C308E1    MFDIV   JMP   :E108     FPT division
039 E00C C312E1    MLOAD   JMP   :E112     Copy operand to accu
040 E00F C31CE1    MSAVE   JMP   :E11C     Copy accu to operand
041 E012 C326E1    MPUT    JMP   :E126     Copy reg A,B,C,D to accu
042 E015 C333E1    MGET    JMP   :E133     Copy accu to reg A,B,C,D
043 E018 C340E1    MFABS   JMP   :E140     FPT ABS
044 E01B C34AE1    MFCHS   JMP   :E14A     FPT change sign accu
045 E01E C343E4    MFINT   JMP   :E443     FPT INT(X)
046 E021 C354E1    MFRAC   JMP   :E154     FPT FRAC
047 E024 C355E8    MPWR    JMP   :E855     FPT power
048 E027 C345E7    MLN     JMP   :E745     LOG
049 E02A C367E6    MEXP    JMP   :E667     EXP
050 E02D C370E8    MLOG    JMP   :E870     LOGT
051 E030 C380E8    MALOG   JMP   :E880     ALOG
052 E033 C3F8E5    MSQRT   JMP   :E5F8     SQR
053 E036 C3D2E7    MSIN    JMP   :E7D2     SIN
054 E039 C3D9E7    MCOS    JMP   :E7D9     COS
055 E03C C394E8    MTAN    JMP   :E894     TAN
056 E03F C36CE9    MASIN   JMP   :E96C     ASIN
057 E042 C3C1E9    MACOS   JMP   :E9C1     ACOS
058 E045 C3ACE8    MATAN   JMP   :E8AC     ATN
059 E048 C314E4    MFIX    JMP   :E414     Change accu to INT
060 E04B C3DEE3    MFLT    JMP   :E3DE     Change accu to FPT
061 E04E C36DE1    MIADD   JMP   :E16D     INT addition
062 E051 C38DE1    MISUB   JMP   :E18D     INT subtraction
063 E054 C3ACE1    MIMUL   JMP   :E1AC     INT multiplication
```

```
064 E057 C32BE2     MIDIV   JMP   :E22B     INT division
065 E05A C338E2     MIREM   JMP   :E238     INT divide remainder
066 E05D C30BE3     MIABS   JMP   :E30B     INT ABS
067 E060 C315E3     MICHS   JMP   :E315     INT change sign accu
068 E063 C32EE3     MIAND   JMP   :E32E     IAND
069 E066 C345E3     MIOR    JMP   :E345     IOR
070 E069 C35CE3     MIXOR   JMP   :E35C     IXOR
071 E06C C373E3     MINOT   JMP   :E373     INOT
072 E06F C3A5E3     MSHL    JMP   :E3A5     SHL
073 E072 C398E3     MSHR    JMP   :E398     SHR
074 E075 C3C0ED     MSA00   JMP   :EDC0     Part of SAVEA
075 E078 C3A6EF     L1E274  JMP   :EFA6     Bank return
076                 *
077                 ************************
078                 * HARDWARE ENTRYPOINTS *
079                 ************************
080                 *
081                 * #00D4 contains offset #7B from #E000 as base
082                 * for HVECA.
083                 *
084 E07B C374E4     HVECA   JMP   :E474     FPT addition
085 E07E C379E4             JMP   :E479     FPT subtraction
086 E081 C37EE4             JMP   :E47E     FPT multiplication
087 E084 C383E4             JMP   :E483     FPT division
088 E087 C388E5             JMP   :E588     Copy operand to accu
089 E08A C399E5             JMP   :E599     Copy accu to operand
090 E08D C35FE5             JMP   :E55F     Copy reg A,B,C,D to accu
091 E090 C36FE5             JMP   :E56F     Copy accu to reg A,B,C,D
092 E093 C388E4             JMP   :E488     FPT ABS
093 E096 C393E4             JMP   :E493     FPT change sign accu
094 E099 C398E4             JMP   :E498     FPT INT(X)
095 E09C C3A0E4             JMP   :E4A0     FPT FRAC
096 E09F C3A1ED             JMP   :EDA1     FPT power
097 E0A2 C3B1E4             JMP   :E4B1     LOG
098 E0A5 C3B6E4             JMP   :E4B6     EXP
099 E0A8 C3BBE4             JMP   :E4BB     LOGT
100 E0AB C3C0E4             JMP   :E4C0     ALOG
101 E0AE C3CCE4             JMP   :E4CC     SQR
102 E0B1 C3D1E4             JMP   :E4D1     SIN
103 E0B4 C3D6E4             JMP   :E4D6     COS
104 E0B7 C3DBE4             JMP   :E4DB     TAN
105 E0BA C3E0E4             JMP   :E4E0     ASIN
106 E0BD C3E5E4             JMP   :E4E5     ACOS
107 E0C0 C3EAE4             JMP   :E4EA     ATN
108 E0C3 C3EFE4             JMP   :E4EF     Change accu to INT
109 E0C6 C39BE4             JMP   :E49B     Change accu to FPT
110 E0C9 C3F4E4             JMP   :E4F4     INT addition
111 E0CC C3F9E4             JMP   :E4F9     INT subtraction
112 E0CF C3FEE4             JMP   :E4FE     INT multiplication
113 E0D2 C303E5             JMP   :E503     INT division
114 E0D5 C308E5             JMP   :E508     INT divide remainder
115 E0D8 C317E5             JMP   :E517     INT ABS
116 E0DB C322E5             JMP   :E522     INT change sign accu
117 E0DE C319ED             JMP   :ED19     IAND
118 E0E1 C326ED             JMP   :ED26     IOR
119 E0E4 C333ED             JMP   :ED33     IXOR
120 E0E7 C343ED             JMP   :ED43     INOT
121 E0EA C36CED             JMP   :ED6C     SHL
122 E0ED C355ED             JMP   :ED55     SHR
123 E0F0 C3C0ED             JMP   :EDC0     Part of SAVEA ) Not via
124 E0F3 C3A6EF             JMP   :EFA6     Bank return   ) AMD9511
125                 *
```

```
126 E0F6 FF                     DATA  :FF
127 E0F7 FF                     DATA  :FF
128 E0F8 FF                     DATA  :FF
129 E0F9 FF                     DATA  :FF
130 E0FA FF                     DATA  :FF
131 E0FB FF                     DATA  :FF
132 E0FC FF                     DATA  :FF
133 E0FD FF                     DATA  :FF
134                     *
135                     ************************
136                     * FPT MULTIPLICATION *
137                     ************************
138                     *
139                     * MACC = MACC * MEM.
140                     *
141                     * Entry: HL: Points to multiplier.
142                     * Exit:  All registers preserved.
143                     *
144 E0FE F5             XFMUL   PUSH  PSW
145 E0FF C5                     PUSH  B
146 E100 D5                     PUSH  D
147 E101 E5                     PUSH  H
148 E102 CD59EA                 CALL  :EA59     FPT multiplication
149 E105 C34DC1                 JMP   :C14D     Popall, ret
150                     *
151                     ****************
152                     * FPT DIVISION *
153                     ****************
154                     *
155                     * MACC = MACC / MEM.
156                     *
157                     * Entry: HL: Points to divisor.
158                     * Exit:  All registers preserved.
159                     *
160 E108 F5             XFDIV   PUSH  PSW
161 E109 C5                     PUSH  B
162 E10A D5                     PUSH  D
163 E10B E5                     PUSH  H
164 E10C CD20EA                 CALL  :EA20     FPT division
165 E10F C34DC1                 JMP   :C14D     Popall, ret
166                     *
167                     ****************************
168                     * COPY OPERAND INTO MACC *
169                     ****************************
170                     *
171                     * Entry: HL: Points to operand.
172                     * Exit:  All registers preserved.
173                     *
174 E112 F5             XLOAD   PUSH  PSW
175 E113 C5                     PUSH  B
176 E114 D5                     PUSH  D
177 E115 E5                     PUSH  H
178 E116 CDFBE9                 CALL  :E9FB     Copy operand in MACC
179 E119 C34DC1                 JMP   :C14D     Popall, ret
180                     *
181                     ************************
182                     * COPY MACC TO OPERAND *
183                     ************************
184                     *
185                     * Entry: HL: Points to operand.
186                     * Exit:  All registers preserved.
```

```
188 E11C F5          XSAVE   PUSH  PSW
189 E11D C5                  PUSH  B
190 E11E D5                  PUSH  D
191 E11F E5                  PUSH  H
192 E120 CDD6E9              CALL  :E9D6     Copy MACC to operand
193 E123 C34DC1              JMP   :C14D     Popall, ret
194                  *
195                  ***************************************
196                  * COPY REGISTERS A,B,C,D INTO MACC *
197                  ***************************************
198                  *
199                  * Entry: None.
200                  * Exit:  All registers preserved.
201                  *
202 E126 E5          XPUT    PUSH  H
203 E127 62                  MOV   H,D       )
204 E128 69                  MOV   L,C       ) DC in HL
205 E129 22D700              SHLD  :00D7     Copy D,C into 00D8/7
206 E12C 60                  MOV   H,B       )
207 E12D 6F                  MOV   L,A       ) BA in HL
208 E12E 22D500              SHLD  :00D5     Copy B,A into 00D6/5
209 E131 E1                  POP   H
210 E132 C9                  RET
211                  *
212                  ***************************************
213                  * COPY MACC INTO REGISTERS A,B,C,D *
214                  ***************************************
215                  *
216                  * Entry: None.
217                  * Exit:  EHLF preserved.
218                  *
219 E133 E5          XGET    PUSH  H
220 E134 2AD700              LHLD  :00D7     Get lobytes MACC
221 E137 4D                  MOV   C,L       )
222 E138 54                  MOV   D,H       ) Into registers C,D
223 E139 2AD500              LHLD  :00D5     Get hibytes MACC
224 E13C 7D                  MOV   A,L       )
225 E13D 44                  MOV   B,H       ) Into registers A,B
226 E13E E1                  POP   H
227 E13F C9                  RET
228                  *
229                  ***********
230                  * FPT ABS *
231                  ***********
232                  *
233                  * For FPT values: MACC = Absolute value
234                  * of MACC.
235                  *
236                  * Entry: None.
237                  * Exit:  All registers preserved.
238                  *
239 E140 F5          XFABS   PUSH  PSW
240 E141 C5                  PUSH  B
241 E142 D5                  PUSH  D
242 E143 E5                  PUSH  H
243 E144 CDEEE9              CALL  :E9EE     Take abs.value of MACC
244 E147 C34DC1              JMP   :C14D     Popall, ret
245                  *
246                  ************************
247                  * FPT: CHANGE SIGN MACC *
248                  ************************
249                  *
```

```
                        * For FPT values: MACC = - MACC
                        *
                        * Entry: None.
                        * Exit:  All registers preserved.
                        *
E14A F5         XFCHS   PUSH  PSW
E14B C5                 PUSH  B
E14C D5                 PUSH  D
E14D E5                 PUSH  H
E14E CDE4E9             CALL  :E9E4     Change sign MACC
E151 C34DC1             JMP   :C14D     Popall, ret
                        *
                        ************
                        * FPT FRAC *
                        ************
                        *
                        * The FPT number in the MACC is replaced by its
                        * fractional part.
                        *
                        * Entry: None.
                        * Exit:  All registers preserved.
                        *
E154 F5         XFRAC   PUSH  PSW
E155 E5                 PUSH  H
E156 CD1EC2             CALL  :C21E     Save MACC on stack
E159 CD43E4             CALL  :E443     Take INT value of MACC
E15C 210000             LXI   H,:0000
E15F 39                 DAD   SP        Pnts to orig MACC on stack
E160 CDB4ED             CALL  :EDB4     Subtract INT(MACC)-MACC
E163 CD4AE1             CALL  :E14A     Make result positive again
E166 33                 INX   SP
E167 33                 INX   SP
E168 33                 INX   SP
E169 33                 INX   SP        Correct SP
E16A E1                 POP   H
E16B F1                 POP   PSW
E16C C9                 RET
                        *
                        ********************
                        * INTEGER ADDITION *
                        ********************
                        *
                        * Signed 32-bit addition: MACC = MACC + MEM.
                        * Evt. overflow handling via C04B.
                        *
                        * Entry: HL: Points to 1st byte of operand.
                        * Exit:  All registers preserved.
                        *
E16D F5         XIADD   PUSH  PSW
E16E C5                 PUSH  B
E16F D5                 PUSH  D
E170 E5                 PUSH  H
E171 CD8CE3             CALL  :E38C     MACC into reg E,B,C,A
                                        D = compl 00D5 EXOR M
E174 D5                 PUSH  D
E175 86                 ADD   M         )
E176 57                 MOV   D,A       )
E177 2B                 DCX   H         )
E178 79                 MOV   A,C       )
E179 8E                 ADC   M         )   Add contents MEM to EBCA
E17A 4F                 MOV   C,A       )   Result in EBCD
E17B 2B                 DCX   H         )
```

```
312 E17C 78                 MOV   A,B       )
313 E17D 8E                 ADC   M         )
314 E17E 47                 MOV   B,A       )
315 E17F 2B                 DCX   H         )
316 E180 7B                 MOV   A,E       )
317 E181 8E                 ADC   M         )
318 E182 5F                 MOV   E,A       )
319 E183 1F                 RAR             Evt carry in msb
320 E184 AB                 XRA   E         Msb=1 if overflow
321 E185 E1                 POP   H         Get compl 00D5 EXOR M
322                                         (0 if different signbits)
323 E186 A4                 ANA   H         Overflow only if different
324                                         signbits
325 E187 FC4BC0     L1E11   CM    :C04B     Then run overflow error
326 E18A C384E3             JMP   :E384     Copy E into A; Then reg
327                                         ABCD into MACC
328                 *
329                 ************************
330                 * INTEGER SUBTRACTION *
331                 ************************
332                 *
333                 * Signed 32-bit subtraction: MACC = MACC - MEM.
334                 * Evt. overflow handling via C04B.
335                 *
336                 * Entry: HL: Points to 1st byte of operand.
337                 * Exit:  All registers preserved.
338                 *
339 E18D F5          XISUB   PUSH  PSW
340 E18E C5                 PUSH  B
341 E18F D5                 PUSH  D
342 E190 E5                 PUSH  H
343 E191 CD8CE3             CALL  :E38C     Copy MACC into reg EBCA
344                                         D = compl 00D5 EXOR M
345 E194 D5                 PUSH  D
346 E195 96                 SUB   M         )
347 E196 57                 MOV   D,A       )
348 E197 2B                 DCX   H         )
349 E198 79                 MOV   A,C       )
350 E199 9E                 SBB   M         )
351 E19A 4F                 MOV   C,A       ) Subtract EBCA - MEM
352 E19B 2B                 DCX   H         )  Result in EBCD
353 E19C 78                 MOV   A,B       )
354 E19D 9E                 SBB   M         )
355 E19E 47                 MOV   B,A       )
356 E19F 2B                 DCX   H         )
357 E1A0 7B                 MOV   A,E       )
358 E1A1 9E                 SBB   M         )
359 E1A2 5F                 MOV   E,A       )
360 E1A3 1F                 RAR             Evt carry in msb
361 E1A4 AB                 XRA   E         Msb=1 if overflow
362 E1A5 E1                 POP   H         Get compl 00D5 EXOR M
363 E1A6 B4                 ORA   H
364 E1A7 2F                 CMA
365 E1A8 B7                 ORA   A         Msb=1 ?
366 E1A9 C387E1             JMP   :E187     Evt run overflow error;
367                                         Copy result into MACC
368                 *
369                 ****************************
370                 * INTEGER MULTIPLICATION *
371                 ****************************
372                 *
373                 * Signed 32-bit multiplication: MACC = MACC * MEM.
```

```
374                     * The overflow exit is taken if both factors are
375                     * more than 2 bytes or the product is longer than
376                     * the signbit of the 4th byte.
377                     * If MEM > 2 bytes, than MACC into DE and MEM into
378                     * MACC, assuming that MACC is max. 2 bytes.
379                     *
380                     * Entry: HL: Points to multiplier.
381                     * Exit:  All registers preserved.
382                     *
383 E1AC F5            XIMUL   PUSH  PSW
384 E1AD C5                    PUSH  B
385 E1AE D5                    PUSH  D
386 E1AF E5                    PUSH  H
387 E1B0 3AD500              LDA   :00D5     Get sign byte
388 E1B3 B7                  ORA   A
389 E1B4 FC15E3              CM    :E315     If nr <0: change sign
390 E1B7 DA25E2              JC    :E225     (Don't work: E315 preserves
391                                          flags; ORA A clears CY)
392 E1BA AE                  XRA   M         Final sign bit in S-flag
393 E1BB F5                  PUSH  PSW       Save it
394 E1BC 7E                  MOV   A,M       )
395 E1BD 23                  INX   H         )
396 E1BE B7                  ORA   A         ) Get MEM into BCDE
397 E1BF 47                  MOV   B,A       ) Set flags on sign byte
398 E1C0 4E                  MOV   C,M       )
399 E1C1 23                  INX   H         )
400 E1C2 56                  MOV   D,M       )
401 E1C3 23                  INX   H         )
402 E1C4 5E                  MOV   E,M       )
403 E1C5 FCC9E3              CM    :E3C9     If MEM <0: negate BCDE
404 E1C8 DA25E2              JC    :E225     Error exit if overflow
405 E1CB B1                  ORA   C
406 E1CC CADFE1              JZ    :E1DF     Jump if MEM <= 2 bytes
407
408                     * MEM > 2 bytes: exchange MACC with BCDE:
409
410 E1CF 2AD500              LHLD  :00D5     Get hibytes MACC
411 E1D2 7C                  MOV   A,H
412 E1D3 B5                  ORA   L
413 E1D4 C225E2              JNZ   :E225     Overflow exit if MACC >
414                                          2 bytes
415 E1D7 2AD700              LHLD  :00D7     Get lobytes MACC in HL
416 E1DA CDCFE3              CALL  :E3CF     Copy reg BCDE (=MEM) into
417                                          MACC
418 E1DD 55                  MOV   D,L       )
419 E1DE 5C                  MOV   E,H       ) Orig. MACC into DE
420
421                     * Now 4-byte nr in MACC and 2 byte nr in DE:
422
423 E1DF 210000      L1E14   LXI   H,:0000
424 E1E2 E5                  PUSH  H         0000 on stack
425 E1E3 21D800              LXI   H,:00D8   Addr lobyte MACC
426 E1E6 4E                  MOV   C,M       lobyte in C
427 E1E7 E3         L1E15    XTHL            On stack: addr current MACC
428                                          byte
429 E1E8 79                  MOV   A,C       Current byte in A
430 E1E9 B7                  ORA   A
431 E1EA CA1EE2              JZ    :E21E     Jump if byte=0
432 E1ED 0680                MVI   B,:80
433 E1EF 79         L1E16    MOV   A,C       ) Current MACC byte in A
434 E1F0 1F                  RAR             )
435 E1F1 4F                  MOV   C,A       )
```

```
436 E1F2 D2F6E1            JNC   :E1F6     ) SHR reg H,L and B as long
437                                         ) no carry from C
438 E1F5 19                DAD   D          ) Else: Add other nr to HL
439 E1F6 7C        L1E17   MOV   A,H        )
440 E1F7 1F                RAR              )
441 E1F8 67                MOV   H,A        )
442 E1F9 7D                MOV   A,L        )-- Effect: Multiply MACC by
443 E1FA 1F                RAR              )           DE, result in B
444 E1FB 6F                MOV   L,A        )
445 E1FC 78                MOV   A,B        )
446 E1FD 1F                RAR              )
447 E1FE 47                MOV   B,A        )
448 E1FF D2EFE1            JNC   :E1EF      ) Do L1E16 max 8 times
449 E202 E3                XTHL             HL pnts to current MACC byte
450 E203 70                MOV   M,B        Result in MACC byte
451 E204 2B                DCX   H          Pnts to next lower MACC byte
452 E205 7D                MOV   A,L        Get lobyte of addr
453 E206 FE04              CPI   :04        Ready?
454 E208 4E                MOV   C,M        Get next lower byte in C
455 E209 C2E7E1            JNZ   :E1E7      Not ready: mult. next byte
456
457                 * Multiplication done:
458
459 E20C E1                POP   H
460 E20D 78                MOV   A,B        Get last result
461 E20E B7                ORA   A
462 E20F FA25E2            JM    :E225      Error exit if overflow
463 E212 7C                MOV   A,H        )
464 E213 B5                ORA   L          ) Check if HL<>0
465 E214 C225E2            JNZ   :E225      Then overflow exit
466 E217 F1        L1E19   POP   PSW        Get final sign in S-flag
467 E218 FC15E3            CM    :E315      Change sign MACC if nr must
468                                         be negative
469 E21B C34DC1            JMP   :C14D      Popall, ret
470
471                 * If MACC byte = 0:
472
473 E21E 45          L1E20   MOV   B,L        ) Move bytes in HLB one byte
474 E21F 6C                MOV   L,H        ) down
475 E220 2600              MVI   H,:00      )
476 E222 C302E2            JMP   :E202      Go to next MACC byte
477
478                 * If overflow error:
479
480 E225 CD4BC0    L1E21   CALL  :C04B      Run overfow error
481 E228 C317E2            JMP   :E217      Quit
482                 *
483                 *
484                 *
485 E22B                   END
```

```
****************************
* S Y M B O L   T A B L E *
****************************

HVECA  E07B   L1E11  E187   L1E14  E1DF   L1E15  E1E7
L1E16  E1EF   L1E17  E1F6   L1E19  E217   L1E20  E21E
L1E21  E225   L1E274 E078   MACOS  E042   MALOG  E030
MASIN  E03F   MATAN  E045   MCOS   E039   MEXP   E02A
MFABS  E018   MFADD  E000   MFCHS  E01B   MFDIV  E009
MFINT  E01E   MFIX   E048   MFLT   E04B   MFMUL  E006
MFRAC  E021   MFSUB  E003   MGET   E015   MIABS  E05D
```

MIADD	E04E	MIAND	E063	MICHS	E060	MIDIV	E057
MIMUL	E054	MINOT	E06C	MIOR	E066	MIREM	E05A
MISUB	E051	MIXOR	E069	MLN	E027	MLOAD	E00C
MLOG	E02D	MPUT	E012	MPWR	E024	MSA00	E075
MSAVE	E00F	MSHL	E06F	MSHR	E072	MSIN	E036
MSQRT	E033	MTAN	E03C	SVECA	E000	XFABS	E140
XFCHS	E14A	XFDIV	E108	XFMUL	E0FE	XFRAC	E154
XGET	E133	XIADD	E16D	XIMUL	E1AC	XISUB	E18D
XLOAD	E112	XPUT	E126	XSAVE	E11C		

```
002                             ORG   :E22B
003                     *
004                     *
005                     *
006                     **********************
007                     * INTEGER DIVISION *
008                     **********************
009                     *
010                     * Signed 32-bit fixed point division:
011                     *       MACC = MACC / MEM.
012                     *
013                     * Entry: HL: Points to divisor.
014                     * Exit:  All registers preserved.
015                     *
016 E22B F5             XIDIV   PUSH  PSW
017 E22C C5                     PUSH  B
018 E22D D5                     PUSH  D
019 E22E E5                     PUSH  H
020 E22F CD42E2                 CALL  :E242     Signed division
021 E232 CDCFE3                 CALL  :E3CF     Quotient into MACC
022 E235 C34DC1                 JMP   :C14D     Popall, ret
023                     *
024                     ***************************
025                     * INT DIVIDE REMAINDER *
026                     ***************************
027                     *
028                     * For INT values: MACC = Remainder of (MACC / MEM).
029                     *
030                     * Entry: HL: Points to divisor.
031                     * Exit:  All registers preserved.
032                     *
033 E238 F5             XIREM   PUSH  PSW
034 E239 C5                     PUSH  B
035 E23A D5                     PUSH  D
036 E23B E5                     PUSH  H
037 E23C CD42E2                 CALL  :E242     Signed division;
038                                             Remainder in MACC
039 E23F C34DC1                 JMP   :C14D     Popall, ret
040                     *
041                     ****************************
042                     * SIGNED INTEGER DIVISION *
043                     ****************************
044                     *
045                     * Divides MACC / MEM; quotient is left in registers
046                     * B,C,D,E and the remainder in MACC.
047                     *
048                     * Entry: HL:      Points to divisor.
049                     *        MACC:    Dividend.
050                     * Exit:  BCDE:    Quotient; remainder in MACC.
051                     *        S-flag:  Set for result.
052                     *        AHL:    Corrupted, CY=0.
053                     *
054 E242 CD8FED         L1E24   CALL  :ED8F     Get signbyte dividend in A,
055                                             compare it with sign divisor
056 E245 78                     MOV   A,B       Get signbyte dividend
057 E246 F5                     PUSH  PSW       Save result compare
058 E247 CD0EEA                 CALL  :EA0E     Copy divisor into BCDE
059 E24A 78                     MOV   A,B       )
060 E24B B1                     ORA   C         ) Check if divisor = 0
061 E24C B2                     ORA   D         )
062 E24D B3                     ORA   E         )
063 E24E CAE4E2                 JZ    :E2E4     Then error 'divide by zero'
```

```
064 E251 78                    MOV   A,B       Get signbyte divisor
065 E252 B7                    ORA   A         Is it negative ?
066 E253 FCC9E3                CM    :E3C9     Then negate divisor
067 E256 DAE4E2                JC    :E2E4     Error exit if overflow
068 E259 C5                    PUSH  B         ) Save divisor on stack
069 E25A D5                    PUSH  D         )
070 E25B CDECE2                CALL  :E2EC     Normalize divisor
071 E25E 2F                    CMA             ) H = pos. value of nr of
072 E25F 3C                    INR   A         ) times shifted for norma-
073 E260 67                    MOV   H,A       ) lisation
074 E261 3AD500                LDA   :00D5     Get signbyte dividend
075 E264 B7                    ORA   A         Is dividend negative ?
076 E265 FC15E3                CM    :E315     Then change sign dividend
077 E268 CDD6E3                CALL  :E3D6     Copy dividend in reg BCDE
078 E26B B1                    ORA   C         )
079 E26C B2                    ORA   D         ) Check if dividend zero
080 E26D B3                    ORA   E         )
081 E26E CAE0E2                JZ    :E2E0     Then abort, leaving 0000 in
082                                            MACC and reg BCDE
083 E271 CDECE2                CALL  :E2EC     Normalize dividend; nrs of
084                                            shifts in A
085 E274 D1                    POP   D         ) Restore divisor in BCDE
086 E275 C1                    POP   B         )
087 E276 84                    ADD   H         Add nr of shifts for divisor
088                                            H is total nrs of shifts
089 E277 FAC9E2                JM    :E2C9     If resulting nrs of shifts
090                                            < 0 then result = 0000
091 E27A CD39EB                CALL  :EB39     Shift divisor left (A) times
092 E27D D5                    PUSH  D         ) Save shifted divisor on
093 E27E C5                    PUSH  B         ) stack
094 E27F 010080                LXI   B,:8000   Init BCDE for max neg number
095 E282 110000                LXI   D,:0000
096 E285 CD55EB                CALL  :EB55     Shift BCDE right (A) times
097 E288 60                    MOV   H,B       ) Shifted BC in HL
098 E289 69                    MOV   L,C       )
099 E28A C1                    POP   B         Get hibytes shifted divisor
100 E28B E3                    XTHL            HL: lobytes shifted divisor;
101                                            shifted BC on stack
102 E28C EB                    XCHG            DE: lobytes shifted divisor;
103                                            HL: shifted DE
104 E28D E5                    PUSH  H         Shifted DE on stack
105 E28E 2AD700        L1E25   LHLD  :00D7     Get lobytes dividend
106 E291 7C                    MOV   A,H       )
107 E292 93                    SUB   E         )
108 E293 67                    MOV   H,A       ) (00D7/8)=(00D7/8)-lobytes
109 E294 7D                    MOV   A,L       ) shifted divisor
110 E295 9A                    SBB   D         )
111 E296 6F                    MOV   L,A       )
112 E297 22D700                SHLD  :00D7     )
113 E29A 2AD500                LHLD  :00D5     Get hibytes dividend
114 E29D 7C                    MOV   A,H       )
115 E29E 99                    SBB   C         )
116 E29F 67                    MOV   H,A       ) (00D5/6)=(00D5/6)-hibytes
117 E2A0 7D                    MOV   A,L       ) shifted divisor
118 E2A1 98                    SBB   B         )
119 E2A2 6F                    MOV   L,A       )
120 E2A3 22D500                SHLD  :00D5     )
121 E2A6 E1            L1E26   POP   H         Get shifted DE
122 E2A7 17                    RAL
123 E2A8 3F                    CMC
124 E2A9 7D                    MOV   A,L
                               RAL
```

```
126 E2AB 6F                 MOV   L,A
127 E2AC 7C                 MOV   A,H
128 E2AD 17                 RAL
129 E2AE 67                 MOV   H,A
130 E2AF E3                 XTHL              Get shifted 'BC' in HL
131 E2B0 7D                 MOV   A,L
132 E2B1 17                 RAL
133 E2B2 6F                 MOV   L,A
134 E2B3 7C                 MOV   A,H
135 E2B4 17                 RAL
136 E2B5 67                 MOV   H,A
137 E2B6 E3                 XTHL              New 'BC' back on stack
138 E2B7 DAD0E2             JC    :E2D0
139 E2BA CD70EB             CALL  :EB70       Rotate BCDE right 1 bit
140 E2BD 7D                 MOV   A,L
141 E2BE 1F                 RAR
142 E2BF E5                 PUSH  H           New 'DE' back on stack
143 E2C0 DA8EE2             JC    :E28E       CY=1: again
144 E2C3 CDF2E2             CALL  :E2F2       MACC = MACC + BCDE
145 E2C6 C3A6E2             JMP   :E2A6       Again
146
147                 * If resulting nr of shifts is negative:
148
149 E2C9 110000      L1E27   LXI   D,:00
150 E2CC D5                 PUSH  D
151 E2CD C3D7E2             JMP   :E2D7       BCDE = 0000; abort
152
153                 * If ready:
154
155 E2D0 7D          L1E28   MOV   A,L
156 E2D1 1F                 RAR
157 E2D2 E5                 PUSH  H
158 E2D3 D4F2E2             CNC   :E2F2       CY=0: MACC = MACC + BCDE
159 E2D6 D1                 POP   D
160 E2D7 C1          L1E29   POP   B
161 E2D8 F1                 POP   PSW         Get result sign compare
162 E2D9 CD88ED             CALL  :ED88       Evt negate BCDE
163 E2DC FC15E3             CM    :E315       Evt change sign MACC
164 E2DF C9                 RET
165
166                 * If dividend is zero:
167
168 E2E0 E1          L1E30   POP   H           ) Save BCDE = 0000
169 E2E1 E1                 POP   H           )
170 E2E2 F1                 POP   PSW
171 E2E3 C9                 RET
172
173                 * If errors:
174
175 E2E4 DC4BC0      L1E31   CC    :C04B       CY=1: Run overflow error
176 E2E7 CC6CC0             CZ    :C06C       Z=1: Run divide by 0 error
177 E2EA F1                 POP   PSW
178 E2EB C9                 RET
179                 *
180                 ***********************************************
181                 * INT: NORMALIZE CONTENTS REGISTERS B,C,D,E *
182                 ***********************************************
183                 *
184                 * Exit: HL preserved.
185                 *
186 E2EC E5          L1E32   PUSH  H
187 E2ED CDA0EB             CALL  :EBA0       Normalize BCDE (INT)
```

```
188 E2F0 E1                 POP   H
189 E2F1 C9                 RET
190                 *
191                 ******************************************
192                 * ADD CONTENTS REGISTERS B,C,D,E TO MACC *
193                 ******************************************
194                 *
195                 * Exit: BCDE preserved, AHL corrupted.
196                 *       F set on hibyte of result.
197                 *
198 E2F2 2AD700    L1E33  LHLD  :00D7      )
199 E2F5 7C              MOV   A,H        )
200 E2F6 83              ADD   E          ) Add DE to 00D7/8
201 E2F7 67              MOV   H;A        )
202 E2F8 7D              MOV   A,L        )
203 E2F9 8A              ADC   D          )
204 E2FA 6F              MOV   L,A
205 E2FB 22D700          SHLD  :00D7
206 E2FE 2AD500          LHLD  :00D5
207 E301 7C              MOV   A,H        )
208 E302 89              ADC   C          )
209 E303 67              MOV   H,A        ) Add BC to 00D5/6
210 E304 7D              MOV   A,L        )
211 E305 88              ADC   B          )
212 E306 6F              MOV   L,A        )
213 E307 22D500          SHLD  :00D5
214 E30A C9              RET
215                 *
216                 ***********
217                 * INT ABS *
218                 ***********
219                 *
220                 * For INT values: MACC = absolute value of MACC.
221                 *
222                 * Exit: All registers preserved.
223                 *
224 E30B F5         XIABS  PUSH  PSW
225 E30C 3AD500            LDA   :00D5     Get sign byte
226 E30F B7                ORA   A
227 E310 FC15E3            CM    :E315     If <0: change sign
228 E313 F1                POP   PSW
229 E314 C9                RET
230                 *
231                 ************************************
232                 * INT: CHANGE SIGN MACC CONTENTS *
233                 ************************************
234                 *
235                 * For INT values: MACC = - MACC.
236                 *
237                 * Exit: All registers preserved.
238                 *
239 E315 F5         XICHS  PUSH  PSW
240 E316 C5                PUSH  B
241 E317 D5                PUSH  D
242 E318 E5                PUSH  H
243 E319 CDD6E3            CALL  :E3D6     Copy MACC into reg BCDE
244 E31C CDC9E3            CALL  :E3C9     Negate BCDE
245 E31F D228E3            JNC   :E328     Jump if no error
246
247                 * If overfow error:
248
249 E322 CD4BC0    L1E36  CALL  :C04B     Run overflow error
```

```
E325 C34DC1            JMP   :C14D     Popall, ret

                     * If O.K.:

E328 CDCFE3     L1E37  CALL  :E3CF     Copy reg BCDE into MACC
E32B C34DC1            JMP   :C14D     Popall, ret
                     *
                     ********
                     * IAND *
                     ********
                     *
                     * Logical 'AND': MACC = MACC IAND MEM.
                     *
                     * Entry: HL points to operand in memory.
                     * Exit:  All registers preserved.
                     *
E32E F5         XIAND  PUSH  PSW
E32F C5                PUSH  B
E330 D5                PUSH  D
E331 E5                PUSH  H
E332 C3E1EC            JMP   :ECE1     Prepare IAND and perform
                     *
                     ****************
                     * PERFORM IAND *
                     ****************
                     *
                     * Performs: MEM IAND EBCA, result in ABCD.
                     *
                     * Entry: HL points to last byte of MEM.
                     *         Fixed point number in EBCA.
                     * Exit:  HL points to 1st byte of MEM.
                     *         E preserved.
                     *
E335 A6         SIAND  ANA   M
E336 57                MOV   D,A
E337 2B                DCX   H
E338 79                MOV   A,C
E339 A6                ANA   M
E33A 4F                MOV   C,A
E33B 2B                DCX   H
E33C 78                MOV   A,B
E33D A6                ANA   M
E33E 47                MOV   B,A
E33F 2B                DCX   H
E340 7B                MOV   A,E
E341 A6                ANA   M
E342 C9                RET
                     *
E343 85E3       L1E40  DBL   :E385     (not used)
                     *
                     *******
                     * IOR *
                     *******
                     *
                     * Logical 'OR': MACC = MACC IOR MEM.
                     *
                     * Entry: HL points to operand in memory.
                     * Exit:  All registers preserved.
                     *
E345 F5         XIOR   PUSH  PSW
E346 C5                PUSH  B
E347 D5                PUSH  D
```

```
312 E348 E5              PUSH  H
313 E349 C3EAEC          JMP   :ECEA     Prepare IOR and perform
314                      *
315                      ***************
316                      * PERFORM IOR *
317                      ***************
318                      *
319                      * Performs MEM IOR EBCA. Result in ABCD.
320                      *
321                      * Entry: HL points to last byte of MEM.
322                      *        Fixed point nr in EBCA.
323                      * Exit:  HL points to 1st byte of MEM.
324                      *        E preserved.
325                      *
326 E34C B6              SIOR    ORA   M
327 E34D 57                      MOV   D,A
328 E34E 2B                      DCX   H
329 E34F 79                      MOV   A,C
330 E350 B6                      ORA   M
331 E351 4F                      MOV   C,A
332 E352 2B                      DCX   H
333 E353 78                      MOV   A,B
334 E354 B6                      ORA   M
335 E355 47                      MOV   B,A
336 E356 2B                      DCX   H
337 E357 7B                      MOV   A,E
338 E358 B6                      ORA   M
339 E359 C9                      RET
340                      *
341 E35A 85E3            L1E43   DBL   :E385     (not used)
342                      *
343                      ********
344                      * IXOR *
345                      ********
346                      *
347                      * Logical 'XOR': MACC = MACC IXOR MEM.
348                      *
349                      * Entry: HL points to operand in memory.
350                      * Exit:  All registers preserved.
351                      *
352 E35C F5              XIXOR   PUSH  PSW
353 E35D C5                      PUSH  B
354 E35E D5                      PUSH  D
355 E35F E5                      PUSH  H
356 E360 C3F3EC                  JMP   :ECF3     Prepare IXOR and perform
357                      *
358                      ****************
359                      * PERFORM IXOR *
360                      ****************
361                      *
362                      * Performs MEM IXOR EBCA, result in ABCD.
363                      *
364                      * Entry: HL points last byte of MEM.
365                      *        Fixed point number in EBCA.
366                      * Exit:  HL points to 1st byte of MEM.
367                      *        E preserved.
368                      *
369 E363 AE              SIXOR   XRA   M
370 E364 57                      MOV   D,A
371 E365 2B                      DCX   H
372 E366 79                      MOV   A,C
                                 XRA   M
```

```
374 E368 4F                 MOV   C,A
375 E369 2B                 DCX   H
376 E36A 78                 MOV   A,B
377 E36B AE                 XRA   M
378 E36C 47                 MOV   B,A
379 E36D 2B                 DCX   H
380 E36E 7B                 MOV   A,E
381 E36F AE                 XRA   M
382 E370 C9                 RET
383                 *
384 E371 85E3       L1E46   DBL   :E385     (not used)
385                 *
386                 ********
387                 * INOT *
388                 ********
389                 *
390                 * Logical 'INOT': MACC = INOT (MACC).
391                 *
392                 * Exit: All registers preserved.
393                 *
394 E373 F5         XINOT   PUSH  PSW
395 E374 C5                 PUSH  B
396 E375 D5                 PUSH  D
397 E376 E5                 PUSH  H
398 E377 CDFCEC             CALL  :ECFC     Copy MACC into ABCD; CMA
399 E37A 5F                 MOV   E,A       Save hibyte
400 E37B 7A                 MOV   A,D
401 E37C 2F                 CMA             INOT D
402 E37D 57                 MOV   D,A
403 E37E 79                 MOV   A,C
404 E37F 2F                 CMA             INOT C
405 E380 4F                 MOV   C,A
406 E381 78                 MOV   A,B
407 E382 2F                 CMA             INOT B
408 E383 47                 MOV   B,A
409 E384 7B         L1E48   MOV   A,E       Get back hibyte
410 E385 CD26E1     L1E49   CALL  :E126     Copy ABCD into MACC
411 E388 C34DC1             JMP   :C14D     Popall, ret
412                 *
413 E38B C9         L1E50   RET             (not used)
414                 *
415                 ************************************
416                 * COPY MACC INTO REG. E,B,C,A      *
417                 * D = compl. 00D5 EXOR hibyte MEM  *
418                 ************************************
419                 *
420                 * Entry: HL points to 1st byte MEM.
421                 * Exit:  HL points to last byte MEM.
422                 *        D = complement EXOR hibytes MACC and MEM.
423                 *            Msb D = 1: sign bits identical.
424                 *            Msb D = 0: sign bits different.
425                 *
426 E38C C301ED     L1E51   JMP   :ED01     Copy MACC into ABCD, A in E:
427                                         jump to E38F
428                 *
429 E38F AE         L1E52   XRA   M         EXOR sign bytes
430 E390 2F                 CMA             Complement result
431 E391 23                 INX   H
432 E392 23                 INX   H
433 E393 23                 INX   H
434 E394 D5                 PUSH  D
435 E395 57                 MOV   D,A       A = compl EXOR signbytes
```

```
E396 F1              POP   PSW       Get D in A
E397 C9              RET
                     *
                     *******
                     * SHR *
                     *******
                     *
                     * MACC = MACC SHR MEM.
                     * Shifts contents MACC right (MEM) places.
                     *
                     * Entry: HL points to operand in memory.
                     * Exit:  All registers preserved.
                     *        Result is 0 if MEM < 0 or > 31.
                     *
E398 F5              XSHR  PUSH  PSW
E399 C5                    PUSH  B
E39A D5                    PUSH  D
E39B E5                    PUSH  H
E39C CD08ED                CALL  :ED08     Check MEM. If <=31:
                                           MACC into BCDE, shift in A
                                           Else: clear ABCDE
E39F CD55EB                CALL  :EB55     Shift BCDE right A places
E3A2 C328E3                JMP   :E328     BCDE into MACC; quit
                     *
                     *******
                     * SHL *
                     *******
                     *
                     * MACC = MACC SHL MEM.
                     * Shifts contents MACC left (MEM) places.
                     *
                     * Entry/exit: See XSHR.
                     *
E3A5 F5              XSHL  PUSH  PSW
E3A6 C5                    PUSH  B
E3A7 D5                    PUSH  D
E3A8 E5                    PUSH  H
E3A9 CD08ED                CALL  :ED08     Check MEM. If <= 31: MACC
                                           into BCDE, shift in A
                                           Else: clear ABCDE
E3AC CD39EB                CALL  :EB39     Shift BCDE left A places
E3AF C328E3                JMP   :E328     BCDE into MACC; quit
                     *
                     ********************************
                     * TEST VALUE OF AN INT NUMBER *
                     ********************************
                     *
                     * Tests if an INT has a value between 0 and 31.
                     * If true: Number into A, else: Clear ABCDE.
                     *
                     * Entry: HL points to a 4-byte number.
                     * Exit:  Nr <= #1F: Number in A.
                     *        Nr > #1F: ABCDE cleared.
                     *        HL points to 4th byte in memory.
                     *
E3B2 7E              XSTST MOV   A,M        Get 1st byte
E3B3 23                    INX   H
E3B4 B6                    ORA   M          OR with 2nd
E3B5 23                    INX   H
E3B6 B6                    ORA   M          OR with 3rd
E3B7 23                    INX   H
E3B8 C2C2E3                JNZ   :E3C2      Jump if highest bytes <>0
```

```
498 E3BB 7E                 MOV   A,M       Get 4th byte
499 E3BC E6E0               ANI   :E0       Test 3 highest bits
500 E3BE 00                 NOP
501 E3BF 00                 NOP
502 E3C0 7E                 MOV   A,M       Get 4th byte in A
503 E3C1 C8                 RZ              Abort if 4th byte <= #1F
504
505                 * If nr > #1F:
506
507 E3C2 3E00        L1E56  MVI   A,:00     )
508 E3C4 47                 MOV   B,A       )
509 E3C5 4F                 MOV   C,A       ) Clear ABCDE
510 E3C6 57                 MOV   D,A       )
511 E3C7 5F                 MOV   E,A       )
512 E3C8 C9                 RET
513                 *
514                 ********************************************
515                 * INT: NEGATE CONTENTS REGISTERS B,C,D,E *
516                 ********************************************
517                 *
518                 * Exit: HL preserved.
519                 *       CY=1: Overflow into msb.
520                 *
521 E3C9 E5          L1E57  PUSH  H
522 E3CA CD82EB               CALL  :EB82     Negate BCDE (INT)
523 E3CD E1                 POP   H
524 E3CE C9                 RET
525                 *
526                 ************************************
527                 * COPY REGISTERS B,C,D,E INTO MACC *
528                 ************************************
529                 *
530                 * Entry: none.
531                 * Exit:  ABCD corrupted, FHL preserved.
532                 *
533 E3CF 78          L1E58  MOV   A,B
534 E3D0 41                 MOV   B,C
535 E3D1 4A                 MOV   C,D
536 E3D2 53                 MOV   D,E
537 E3D3 C326E1             JMP   :E126     Copy ABCD into MACC
538                 *
539                 ************************************
540                 * COPY MACC INTO REGISTERS B,C,D,E *
541                 ************************************
542                 *
543                 * Entry: None.
544                 * Exit:  AFHL preserved.
545                 *
546 E3D6 F5          L1E59  PUSH  PSW
547 E3D7 CD33E1               CALL  :E133     Copy MACC into ABCD
548 E3DA C313ED               JMP   :ED13     Copy ABCD into BCDE
549                 *
550 E3DD C9          L1E60  RET
551                 *
552                 ********************************
553                 * CHANGE CONTENTS MACC TO FPT *
554                 ********************************
555                 *
556                 * Result is incorrect if MACC = 80 00 00 00.
557                 * Then exponent is 1 too high (E3FC should be
558                 * a NOP instruction).
559                 *
```

```
560                     * Entry: None.
561                     * Exit:  All registers preserved.
562                     *
563 E3DE F5         XFLT    PUSH  PSW
564 E3DF C5                 PUSH  B
565 E3E0 D5                 PUSH  D
566 E3E1 E5                 PUSH  H
567 E3E2 CDD6E3             CALL  :E3D6     Copy MACC into BCDE
568 E3E5 CD83ED             CALL  :ED83     Check if BCDE is 0.
569 E3E8 CA0EE4             JZ    :E40E     Then clear MACC + BCDE
570 E3EB 2620               MVI   H,:20     Init exp.byte for pos.nr
571 E3ED 78                 MOV   A,B       )
572 E3EE B7                 ORA   A         ) Check sign bit
573 E3EF F2FDE3             JP    :E3FD     Jump if nr is positive
574
575                     * If INT nr is negative:
576
577 E3F2 26A0               MVI   H,:A0     Init exp.byte for neg.nr
578 E3F4 CDC9E3             CALL  :E3C9     Negate BCDE
579 E3F7 D2FDE3             JNC   :E3FD     Jump if no overflow
580 E3FA 0680               MVI   B,:80
581 E3FC 24                 INR   H
582
583                     * Convert to FPT:
584
585 E3FD 7C           L1E62  MOV   A,H       Get init.exp.byte
586 E3FE 21D500             LXI   H,:00D5   Addr MACC
587 E401 77                 MOV   M,A       Init.exp.byte in MACC
588 E402 E5                 PUSH  H
589 E403 CD96EB             CALL  :EB96     Normalize BCDE
590 E406 E1                 POP   H
591 E407 7E                 MOV   A,M       Get init.exp.byte
592 E408 CDDBE9             CALL  :E9DB     Copy ABCD into MACC
593 E40B C34DC1             JMP   :C14D     Popall, ret
594
595                     * If INT number is 0:
596
597 E40E CD16EA       L1E63  CALL  :EA16     Clear MACC + reg ABCD
598 E411 C34DC1             JMP   :C14D     Popall; ret
599                     *
600                     *
601                     *
602 E414                    END
```

```
*****************************
* S Y M B O L   T A B L E *
*****************************

L1E24  E242   L1E25  E28E   L1E26  E2A6   L1E27  E2C9
L1E28  E2D0   L1E29  E2D7   L1E30  E2E0   L1E31  E2E4
L1E32  E2EC   L1E33  E2F2   L1E36  E322   L1E37  E328
L1E40  E343   L1E43  E35A   L1E46  E371   L1E48  E384
L1E49  E385   L1E50  E38B   L1E51  E38C   L1E52  E38F
L1E56  E3C2   L1E57  E3C9   L1E58  E3CF   L1E59  E3D6
L1E60  E3DD   L1E62  E3FD   L1E63  E40E   SIAND  E335
SIOR   E34C   SIXOR  E363   XFLT   E3DE   XIABS  E30B
XIAND  E32E   XICHS  E315   XIDIV  E22B   XINOT  E373
XIOR   E345   XIREM  E238   XIXOR  E35C   XSHL   E3A5
XSHR   E398   XSTST  E3B2
```

```
002                             ORG   :E414
003                     *
004                     *
005                     *
006                     ************************************
007                     * CHANGE CONTENTS MACC TO INTEGER *
008                     ************************************
009                     *
010                     * Entry: None.
011                     * Exit:  All registers preserved.
012                     *
013 E414 F5             XFIX    PUSH  PSW
014 E415 C5                     PUSH  B
015 E416 D5                     PUSH  D
016 E417 E5                     PUSH  H
017 E418 CD33E1                 CALL  :E133     Copy MACC to ABCD
018 E41B F5                     PUSH  PSW       Save exp.byte
019 E41C E67F                   ANI   :7F       Exponent only
020 E41E CA3BE4                 JZ    :E43B     Exp=0: Clear MACC, abort
021 E421 FE40                   CPI   :40       Exp negative ?
022 E423 D23BE4                 JNC   :E43B     Then clear MACC, abort
023 E426 D620                   SUI   :20       Exp >= 32 ?
024 E428 D23FE4                 JNC   :E43F     Then run overflow error
025 E42B 2F                     CMA             ) 2-complement of exp
026 E42C 3C                     INR   A         )
027 E42D CD55EB                 CALL  :EB55     Shift BCDE right A times
028 E430 F1                     POP   PSW       Get exp byte
029 E431 B7                     ORA   A
030 E432 FCC9E3                 CM    :E3C9     Nr <0: negate BCDE
031 E435 DA22E3                 JC    :E322     Jump if overflow
032 E438 C328E3                 JMP   :E328     Copy BCDE into MACC, abort
033
034                     * If number is 0 or exponent negative:
035
036 E43B F1             L1E65   POP   PSW
037 E43C C30EE4                 JMP   :E40E     Clear MACC, abort
038
039                     * if overflow:
040
041 E43F F1             L1E66   POP   PSW
042 E440 C322E3                 JMP   :E322     Run overflow error, abort
043                     *
044                     **************
045                     * FPT INT(X) *
046                     **************
047                     *
048                     * The contents of the MACC is replaced by its
049                     * FPT integer part. The fractional bits of the
050                     * mantissa are masked off.
051                     *
052                     * Exit: All registers preserved.
053                     *
054 E443 F5             XFINT   PUSH  PSW
055 E444 C5                     PUSH  B
056 E445 D5                     PUSH  D
057 E446 E5                     PUSH  H
058 E447 21D500                 LXI   H,:00D5   Addr MACC
059 E44A 7E                     MOV   A,M       Get exp.byte
060 E44B E67F                   ANI   :7F       Exponent only
061 E44D CA0EE4                 JZ    :E40E     If exp=0: Clear MACC, abort
062 E450 FE40                   CPI   :40       Exp negative ?
063 E452 D20EE4                 JNC   :E40E     Then clear MACC, abort
```

```
064 E455 D619              SUI   :19       Exp -  nr of mantissa bits
065 E457 21D800            LXI   H,:00D8    Addr lobyte MACC
066 E45A 1E03              MVI   E,:03      3 bytes in mantissa
067 E45C 57                MOV   D,A        Rest exp in D
068 E45D 0E08       L1E68  MVI   C,:08      8 bits pro byte
069 E45F 14         L1E69  INR   D          Rest exp + 1
070 E460 37                STC              Mask '1' for bits reqd
071 E461 F265E4            JP    :E465      If rest exp >=0
072 E464 3F                CMC              Mask '0' for bits not reqd
073 E465 1F         L1E70  RAR              Shift CY into A to make mask
074 E466 0D                DCR   C
075 E467 C25FE4            JNZ   :E45F      Next bit if not ready
076 E46A A6                ANA   M          Mask MACC byte with mask
077 E46B 77                MOV   M,A        Result back in MACC
078 E46C 2B                DCX   H          Addr next MACC byte
079 E46D 1D                DCR   E
080 E46E C25DE4            JNZ   :E45D      Next byte if not ready
081 E471 C34DC1            JMP   :C14D      Popall, ret
082                  *
083                  **********************
084                  * AMD: FPT ADDITION *
085                  **********************
086                  *
087                  * MTOS = MTOS + MEM.
088                  *
089                  * Entry: HL points to operand in memory.
090                  * Exit:  All registers preserved.
091                  *
092 E474 CD27E5      ZFADD  CALL  :E527      Wait, load, operate imm.
093 E477 10                DATA  :10        FPT addition
094 E478 C9                RET
095                  *
096                  **************************
097                  * AMD: FPT SUBTRACTION *
098                  **************************
099                  *
100                  * MTOS = MTOS - MEM.
101                  *
102                  * Entry: HL points to operand in memory.
103                  * Exit:  All registers preserved.
104                  *
105 E479 CD27E5      ZFSUB  CALL  :E527      Wait, load, operate imm.
106 E47C 11                DATA  :11        FPT subtraction
107 E47D C9                RET
108                  *
109                  ****************************
110                  * AMD: FPT MULTIPLICATION *
111                  ****************************
112                  *
113                  * MTOS = MTOS * MEM.
114                  *
115                  * Entry: HL points to multiplier in memory.
116                  * Exit:  All registers preserved.
117                  *
118 E47E CD27E5      ZFMUL  CALL  :E527      Wait, load, operate imm.
119 E481 12                DATA  :12        FPT multiplication
120 E482 C9                RET
121                  *
122                  **********************
123                  * AMD: FPT DIVISION *
124                  **********************
125                  *
```

```
126                     * MTOS = MTOS / MEM.
127                     *
128                     * Entry: HL points to divisor in memory.
129                     * Exit:  All registers preserved.
130                     *
131 E483 CD27E5         ZFDIV   CALL  :E527     Wait, load, operate imm.
132 E486 13                    DATA  :13       FPT division
133 E487 C9                    RET
134                     *
135                     ****************
136                     * AMD: FPT ABS *
137                     ****************
138                     *
139                     * MTOS is replaced by its absolute value (FPT).
140                     *
141                     * Entry: None.
142                     * Exit:  All registers preserved.
143                     *
144 E488 F5             ZFABS   PUSH  PSW
145 E489 CD95ED                 CALL  :ED95     Get status bits
146 E48C 00                    NOP
147 E48D 87                    ADD   A         Test bit 6 (sign)
148 E48E FC93E4                 CM    :E493     Change sign if reqd (FPT)
149 E491 F1                    POP   PSW
150 E492 C9                    RET
151                     *
152                     ****************************************
153                     * AMD: CHANGE SIGN MTOS CONTENTS (FPT) *
154                     ****************************************
155                     *
156                     * MTOS = - MTOS.
157                     *
158                     * Entry: None.
159                     * Exit:  All registers preserved.
160                     *
161 E493 CD35E5         ZFCHS   CALL  :E535     Wait, operate immediate
162 E496 15                    DATA  :15       FPT change sign MTOS
163 E497 C9                    RET
164                     *
165                     ******************
166                     * AMD: FPT INT(X) *
167                     ******************
168                     *
169                     * MTOS is replaced by its integer part.
170                     *
171 E498 CDEFE4         ZFINT   CALL  :E4EF     Convert MTOS to INT
172
173                     * Entry for AMD: Change MTOS to FPT:
174
175 E49B CD35E5         ZFLT    CALL  :E535     Wait, load, operate imm.
176 E49E 12                    DATA  :12       Convert MTOS to FPT
177 E49F C9                    RET
178                     *
179                     ****************
180                     * AMD: FPT FRAC *
181                     ****************
182                     *
183                     * MTOS is replaced by its fractional part.
184                     *
185 E4A0 CD35E5         ZFRAC   CALL  :E535     Wait, load, operate imm.
186 E4A3 37                    DATA  :37       Push MTOS
187 E4A4 CD98E4                CALL  :E498     MTOS = INT(MTOS)
```

```
188 E4A7 CD35E5             CALL  :E535     Wait, load, operate imm.
189 E4AA 11                 DATA  :11       Subtract whole number
190 E4AB C9                 RET
191                 *
192                 *********************************
193                 * part of AMD: POWER (1EDA1) *
194                 *********************************
195                 *
196                 * Entry: HL points to operand in memory.
197                 * Exit:  All registers preserved.
198                 *
199 E4AC CD27E5     MPR14   CALL  :E527     Wait, load, operate imm.
200 E4AF 0B                 DATA  :0B       MTOS = MTOS ^ MEM
201 E4B0 C9                 RET
202                 *
203                 ************
204                 * AMD: LOG *
205                 ************
206                 *
207 E4B1 CD35E5     ZLN     CALL  :E535     Wait, operate immediate
208 E4B4 09                 DATA  :09       MTOS = LN (MTOS)
209 E4B5 C9                 RET
210                 *
211                 ************
212                 * AMD: EXP *
213                 ************
214                 *
215 E4B6 CD35E5     ZEXP    CALL  :E535     Wait, operate immediate
216 E4B9 0A                 DATA  :0A       MTOS = E ^ MTOS
217 E4BA C9                 RET
218                 *
219                 *************
220                 * AMD: LOGT *
221                 *************
222                 *
223 E4BB CD35E5     ZLOG    CALL  :E535     Wait, operate immediate
224 E4BE 08                 DATA  :08       MTOS = LOG (MTOS)
225 E4BF C9                 RET
226                 *
227                 *************
228                 * AMD: ALOG *
229                 *************
230                 *
231 E4C0 E5         ZALOG   PUSH  H
232 E4C1 2190E8             LXI   H,:E890   Addr 1/logn(10)
233 E4C4 CD83E4             CALL  :E483     MTOS = MTOS/MEM
234 E4C7 E1                 POP   H
235 E4C8 CDB6E4             CALL  :E4B6     MTOS = e ^ MTOS
236 E4CB C9                 RET
237                 *
238                 ************
239                 * AMD: SQR *
240                 ************
241                 *
242 E4CC CD35E5     ZSQRT   CALL  :E535     Wait, operate immediate
243 E4CF 1C                 DATA  :1C       MTOS = SQRT (MTOS)
244 E4D0 C9                 RET
245                 *
246                 ************
247                 * AMD: SIN *
248                 ************
```

```
E4D1 CD35E5      ZSIN    CALL  :E535     Wait, operate immediate
E4D4 02                  DATA  :02       MTOS = SIN (MTOS)
E4D5 C9                  RET
                 *
                 ************
                 * AMD: COS *
                 ************
                 *
E4D6 CD35E5      ZCOS    CALL  :E535     Wait, operate immediate
E4D9 03                  DATA  :03       MTOS = COS (MTOS)
E4DA C9                  RET
                 *
                 ************
                 * AMD: TAN *
                 ************
                 *
E4DB CD35E5      ZTAN    CALL  :E535     Wait, operate immediate
E4DE 04                  DATA  :04       MTOS = TAN (MTOS)
E4DF C9                  RET
                 *
                 *************
                 * AMD: ASIN *
                 *************
                 *
E4E0 CD35E5      ZASIN   CALL  :E535     Wait, operate immediate
E4E3 05                  DATA  :05       MTOS = ASIN (MTOS)
E4E4 C9                  RET
                 *
                 *************
                 * AMD: ACOS *
                 *************
                 *
E4E5 CD35E5      ZACOS   CALL  :E535     Wait, operate immediate
E4E8 06                  DATA  :06       MTOS = ACOS (MTOS)
E4E9 C9                  RET
                 *
                 ************
                 * AMD: ATN *
                 ************
                 *
E4EA CD35E5      ZATAN   CALL  :E535     Wait, operate immediate
E4ED 07                  DATA  :07       MTOS = ATAN (MTOS)
E4EE C9                  RET
                 *
                 ****************************************
                 * AMD: CHANGE CONTENTS MTOS TO INTEGER *
                 ****************************************
                 *
E4EF CD35E5      ZFIX    CALL  :E535     Wait, operate immediate
E4F2 1E                  DATA  :1E       MTOS = INT(MTOS)
E4F3 C9                  RET
                 *
                 **********************
                 * AMD: INT ADDITION *
                 **********************
                 *
                 * MTOS = MTOS + MEM.
                 *
                 * Entry: HL points to number in memory.
                 * Exit:  All registers reserved.
                 *
E4F4 CD27E5      ZIADD   CALL  :E527     Wait, load, operate imm.
```

```
312 E4F7 2C                DATA  :2C       INT addition
313 E4F8 C9                RET
314                *
315                ************************
316                * AMD: INT SUBTRACTION *
317                ************************
318                *
319                * MTOS = MTOS - MEM.
320                *
321                * Entry: HL points to number in memory.
322                * Exit:  All registers preserved.
323                *
324 E4F9 CD27E5    ZISUB   CALL  :E527     Wait, load, operate imm.
325 E4FC 2D                 DATA  :2D       INT subtraction
326 E4FD C9                RET
327                *
328                ****************************
329                * AMD: INT MULTIPLICATION *
330                ****************************
331                *
332                * MTOS = MTOS * MEM.
333                *
334                * Entry: HL points to number in memory.
335                * Exit:  All registers preserved.
336                *
337 E4FE CD27E5    ZIMUL   CALL  :E527     Wait, load, operate imm.
338 E501 2E                 DATA  :2E       INT multiplication
339 E502 C9                RET
340                *
341                **********************
342                * AMD: INT DIVISION *
343                **********************
344                *
345                * MTOS = MTOS / MEM.
346                *
347                * Entry: HL points to number in memory.
348                * Exit:  All registers preserved.
349                *
350 E503 CD27E5    ZIDIV   CALL  :E527     Wait, load, operate imm.
351 E506 2F                 DATA  :2F       INT division
352 E507 C9                RET
353                *
354                ******************************
355                * AMD: INT DIVIDE REMAINDER *
356                ******************************
357                *
358                * MTOS = INT remainder of MTOS / MEM.
359                *
360                * Entry: HL points to number in memory.
361                * Exit:  All registers preserved.
362                *
363 E508 CD35E5    ZIREM   CALL  :E535     Wait, operate immediate
364 E50B 37                 DATA  :37       Push MTOS
365 E50C CD03E5            CALL  :E503     INT divide
366 E50F CDFEE4            CALL  :E4FE     INT multiply back
367 E512 CD35E5            CALL  :E535     Wait, operate immediate
368 E515 2D                 DATA  :2D       Subtract: difference =
369                                         remainder
370 E516 C9                RET
371                *
372                *
373                *
```

```
                        ****************
                        * AMD: INT ABS *
                        ****************
                        *
                        * MTOS = absolute value of MTOS (INT).
                        *
E517 F5         ZIABS   PUSH  PSW
E518 CD95ED             CALL  :ED95     Get status bits
E51B 00                 NOP
E51C 87                 ADD   A         Test bit 6 (sign)
E51D FC22E5             CM    :E522     Change sign MTOS if reqd
E520 F1                 POP   PSW
E521 C9                 RET
                        *
                        ********************************
                        * AMD: CHANGE SIGN MTOS (INT) *
                        ********************************
                        *
E522 CD35E5     ZICHS   CALL  :E535     Wait, operate immediate
E525 34                 DATA  :34       MTOS = - MTOS
E526 C9                 RET
                        *
                        ***************************************
                        * AMD: WAIT, LOAD, OPERATE IMMEDIATE *
                        ***************************************
                        *
                        * Call has to be followed by a 1 byte AMD command.
                        *
E527 CD3BE5     WLOPI   CALL  :E53B     Wait for ready, evt. error
                                        indications
E52A CD88E5             CALL  :E588     Load 2nd operand in MTOS
E52D E3         OPI     XTHL            HL pnts to command byte
E52E F5                 PUSH  PSW
E52F CDCCEC             CALL  :ECCC     Issue command to AMD
E532 F1                 POP   PSW
E533 E3                 XTHL            Restore returnaddr
E534 C9                 RET
                        *
                        ********************************
                        * AMD: WAIT, OPERATE IMMEDIATE *
                        ********************************
                        *
                        * Call has to be followed by a 1 byte AMD command.
                        *
E535 CD3BE5     WOPI    CALL  :E53B     Wait ready, evt. error
                                        indications
E538 C32DE5             JMP   :E52D     Issue command to AMD
                        *
                        ***********************************
                        * AMD: WAIT FOR MATH.CHIP READY *
                        ***********************************
                        *
                        * Waits for math.chip ready. Handles eventual
                        * errors if found.
                        *
                        * Exit: If no errors: All registers preserved.
                        *
E53B F5         WMATH   PUSH  PSW
E53C 3A02FB     WMT10   LDA   :FB02     Get status math.chip
E53F B7                 ORA   A
E540 FA3CE5             JM    :E53C     If busy: wait for ready
E543 E61E               ANI   :1E       Error codes only
```

```
436 E545 CA5DE5             JZ    :E55D     Abort if no errors
437 E548 00                 NOP
438 E549 CDD2EC             CALL  :ECD2     Reset AMD status
439 E54C 1F                 RAR
440 E54D 1F                 RAR
441 E54E DC4BC0             CC    :C04B     Evt. run overflow error
442 E551 1F                 RAR
443 E552 DC65C0             CC    :C065     Evt. run underflow error
444 E555 1F                 RAR
445 E556 DC5EC0             CC    :C05E     Evt. run argument error
446 E559 1F                 RAR
447 E55A DC6CC0             CC    :C06C     Evt. run divide by 0 error
448 E55D F1           WMT20 POP   PSW       Normal return
449 E55E C9                 RET
450                         *
451                         ******************************************
452                         * AMD: COPY REGISTERS A,B,C,D INTO MTOS *
453                         ******************************************
454                         *
455 E55F F5           ZPUT  PUSH  PSW
456 E560 7A                MOV   A,D       D into A
457 E561 CD3BE5             CALL  :E53B     Wait ready
458 E564 3200FB       ZPT10 STA   :FB00     Copy (D) into MTOS
459 E567 79                 MOV   A,C       C into A
460 E568 3200FB             STA   :FB00     Copy (C) into MTOS
461 E56B 78                 MOV   A,B
462 E56C C3D9EC             JMP   :ECD9     Copy (B)+(A) into MTOS
463                         *
464                         ******************************************
465                         * AMD: COPY MTOS INTO REGISTERS A,B,C,D *
466                         ******************************************
467                         *
468 E56F CD3BE5       ZGET  CALL  :E53B     Wait ready
469 E572 3A00FB             LDA   :FB00     Get hibyte from MTOS
470 E575 F5                 PUSH  PSW       Save it on stack
471 E576 3A00FB             LDA   :FB00     Get 2nd byte from MTOS
472 E579 47                 MOV   B,A       Store it in B
473 E57A 3A00FB             LDA   :FB00     Get 3rd byte from MTOS
474 E57D 4F                 MOV   C,A       Store it in C
475 E57E 3A00FB             LDA   :FB00     Get lobyte from MTOS
476 E581 57                 MOV   D,A       Store it in D
477 E582 C364E5             JMP   :E564     Restore MTOS
478                         *
479                         ***************
480                         * (not used) *
481                         ***************
482                         *
483 E585 55           L1E109 MOV    D,L
484 E586 E1                 POP   H
485 E587 C9                 RET
486                         *
487                         ********************************
488                         * AMD: COPY OPERAND INTO MTOS *
489                         ********************************
490                         *
491                         * Entry: HL points to 1st byte of operand.
492                         * Exit:  All registers preserved.
493                         *
494 E588 F5           ZLOAD PUSH  PSW
495 E589 C5                 PUSH  B
496 E58A D5                 PUSH  D
                            PUSH  H
```

```
498 E58C 7E                 MOV   A,M       1st byte in A
499 E58D 23                 INX   H
500 E58E 46                 MOV   B,M       2nd byte in B
501 E58F 23                 INX   H
502 E590 4E                 MOV   C,M       3rd byte in C
503 E591 23                 INX   H
504 E592 56                 MOV   D,M       4th byte in D
505 E593 CD5FE5             CALL  :E55F     Copy ABCD into MTOS
506 E596 C34DC1             JMP   :C14D     Popall, ret
507                 *
508                 ******************************
509                 * AMD: COPY MTOS TO OPERAND *
510                 ******************************
511                 *
512                 * Entry: HL points to 1st byte of operand.
513                 * Exit:  All registers preserved.
514                 *
515 E599 F5          ZSAVE   PUSH  PSW
516 E59A C5                  PUSH  B
517 E59B D5                  PUSH  D
518 E59C E5                  PUSH  H
519 E59D CD6FE5              CALL  :E56F     Copy MTOS into ABCD
520 E5A0 77                  MOV   M,A       )
521 E5A1 23                  INX   H         )
522 E5A2 70                  MOV   M,B       )
523 E5A3 23                  INX   H         ) Copy ABCD into operand
524 E5A4 71                  MOV   M,C       )
525 E5A5 23                  INX   H         )
526 E5A6 72                  MOV   M,D       )
527 E5A7 C34DC1              JMP   :C14D     Popall, ret
528                 *
529                 ************************
530                 * CALCULATE TAYLOR SUM *
531                 ************************
532                 *
533                 * Entry: HL points to a list with FPT constants
534                 *         (Mi with i = 0,1,2,..).
535                 *        MACC: Const. term of Taylor series (S0).
536                 *        00E3-E6: Initial power of argument (P).
537                 *        00E7-EA: Argument X.
538                 *
539                 * Routine computes:
540                 * Sum = S0 + M0*P + M1*P*X + M2*P*X^2 + ......
541                 *
542                 * Exit: 00EB-EE: Sum of series.
543                 *       00E3-E6: Last P*X^i added in.
544                 *       00E7-EA: Preserved.
545                 *       MACC + ABCD: Sum of series.
546                 *       FEHL corrupted.
547                 *
548 E5AA E5          L1E112  PUSH  H         Save table pntr
549 E5AB 21EB00            LXI   H,:00EB
550 E5AE CDD6E9            CALL  :E9D6     Copy MACC (S0) into 00EB-EE
551 E5B1 21E300            LXI   H,:00E3
552 E5B4 CDFBE9            CALL  :E9FB     Copy 00E3-E6 (P) into MACC
553 E5B7 E1          L1E113  POP   H         )
554 E5B8 E5                  PUSH  H         ) Get and save table pntr
555 E5B9 CD59EA              CALL  :EA59     MACC = P * Mi
556 E5BC 21EB00              LXI   H,:00EB
557 E5BF E5                  PUSH  H
558 E5C0 CD72EA              CALL  :EA72     MACC = sum + P * Mi
559 E5C3 E1                  POP   H
```

```
560 E5C4 CDDBE9              CALL  :E9DB     Result in 00EB-EE
561 E5C7 3ADE00              LDA   :00DE     Get difference in exp
562 E5CA B7                  ORA   A
563 E5CB F2D3E5              JP    :E5D3
564 E5CE FEE8                CPI   :E8
565 E5D0 DAF3E5              JC    :E5F3
566 E5D3 E1          L1E114  POP   H         Get table pntr
567 E5D4 23                  INX   H
568 E5D5 23                  INX   H
569 E5D6 23                  INX   H
570 E5D7 23                  INX   H         HL pnts to next table entry
571 E5D8 E5                  PUSH  H         Save table pntr
572 E5D9 23                  INX   H
573 E5DA 7E                  MOV   A;M
574 E5DB B7                  ORA   A
575 E5DC F2F3E5              JP    :E5F3     Jump if ready
576 E5DF 21E300              LXI   H,:00E3
577 E5E2 E5                  PUSH  H
578 E5E3 CDFBE9              CALL  :E9FB     Copy 00E3-E6 into MACC
579                                          and reg ABCD
580 E5E6 21E700              LXI   H,:00E7
581 E5E9 CD59EA              CALL  :EA59     Multiply with 00E7-EA
582 E5EC E1                  POP   H         HL=00E3
583 E5ED CDDBE9              CALL  :E9DB     Copy ABCD into 00E3-E6
584 E5F0 C3B7E5              JMP   :E5B7     Calc next sum
585                  *
586 E5F3 E1          L1E115  POP   H
587 E5F4 CDF8E9              CALL  :E9F8     Copy result into MACC
588                                          and reg ABCD
589 E5F7 C9                  RET
590                  *
591                  *
592                  *
593 E5F8                     END
```

****
* S Y M B O L T A B L E *
****

L1E109	E585	L1E112	E5AA	L1E113	E5B7	L1E114	E5D3
L1E115	E5F3	L1E65	E43B	L1E66	E43F	L1E68	E45D
L1E69	E45F	L1E70	E465	MPR14	E4AC	OPI	E52D
WLOPI	E527	WMATH	E53B	WMT10	E53C	WMT20	E55D
WOPI	E535	XFINT	E443	XFIX	E414	ZACOS	E4E5
ZALOG	E4C0	ZASIN	E4E0	ZATAN	E4EA	ZCOS	E4D6
ZEXP	E4B6	ZFABS	E488	ZFADD	E474	ZFCHS	E493
ZFDIV	E483	ZFINT	E498	ZFIX	E4EF	ZFLT	E49B
ZFMUL	E47E	ZFRAC	E4A0	ZFSUB	E479	ZGET	E56F
ZIABS	E517	ZIADD	E4F4	ZICHS	E522	ZIDIV	E503
ZIMUL	E4FE	ZIREM	E508	ZISUB	E4F9	ZLN	E4B1
ZLOAD	E588	ZLOG	E4BB	ZPT10	E564	ZPUT	E55F
ZSAVE	E599	ZSIN	E4D1	ZSQRT	E4CC	ZTAN	E4DB

```
002                             ORG   :E5F8
003                     *
004                     *
005                     *
006                     ************
007                     * FPT SQRT *
008                     ************
009                     *
010                     * MACC = SQRT (MACC).
011                     *
012                     * Method: approximation followed by Newton
013                     * iterations.
014                     *
015                     * Let X= 2^(2K)*F. Then 2^(2K) is exponent and F
016                     * is mantissa.
017                     *
018                     * Then SQRT(X)=2^K*SQRT(F). 2^K is exp/2.
019                     *       SQRT(F)=P(i):
020                     *           1st approx: P(1)=a*F+b.
021                     *                       0.5<=F<1: values a1 and b1.
022                     *                         1<=F<2: values a2 and b2.
023                     *           Iterations: P(i+1)=(P(i)+F/P(i))/2.
024                     *       Final SQRT(F): P(3).
025                     *
026                     * Exit: All registers preserved.
027                     *
028 E5F8 F5             XSQRT   PUSH  PSW
029 E5F9 C5                     PUSH  B
030 E5FA D5                     PUSH  D
031 E5FB E5                     PUSH  H
032 E5FC CDF1EB                 CALL  :EBF1     Exp.byte MACC in A (2K)
033 E5FF CA3AE6                 JZ    :E63A     Abort if MACC=0
034 E602 07                     RLC
035 E603 DAD0E9                 JC    :E9D0     Run argument error if nr
036                                               in MACC is negative
037 E606 07                     RLC
038 E607 0F                     RRC
039 E608 1F                     RAR
040 E609 B7                     ORA   A
041 E60A 1F                     RAR             A is exp/2 (K)
042 E60B F5                     PUSH  PSW       Save it
043 E60C 3E00                   MVI   A,:00     Set A=0 if lsb exp =0
044 E60E 115FE6                 LXI   D,:E65F   Addr a1,b1 for 0.5<=F<1
045 E611 D218E6                 JNC   :E618
046 E614 3C                     INR   A         Set A=1 if lsb exp =1
047 E615 1157E6                 LXI   D,:E657   Addr a2,b2 for 1<=F<2
048 E618 77             L1E117  MOV   M,A       Init exp byte MACC
049 E619 E5                     PUSH  H         Save addr MACC
050 E61A 21E300                 LXI   H,:00E3
051 E61D CD1CE1                 CALL  :E11C     Copy MACC (F) into 00E3-E6
052 E620 EB                     XCHG
053 E621 E5                     PUSH  H         Save addr a/b
054 E622 CD59EA                 CALL  :EA59     Calc a*F
055 E625 E1                     POP   H
056 E626 23                     INX   H
057 E627 23                     INX   H
058 E628 23                     INX   H
059 E629 23                     INX   H         Pnts to b
060 E62A CD72EA                 CALL  :EA72     Calc P(1)=a*F+b
061 E62D CD3DE6                 CALL  :E63D     Calc P(2)
062 E630 CD3DE6                 CALL  :E63D     Calc P(3); result in MACC
063                                               and reg ABCD
```

```
064 E633 E1                 POP   H          Get addr MACC
065 E634 C1                 POP   B          Get exp/2 (K) in B
066 E635 80                 ADD   B          Add it to exp SQRT(F)
067 E636 E67F               ANI   :7F        Result must be positive
068 E638 77                 MOV   M,A        Final exp.byte into MACC
069 E639 00                 NOP
070 E63A C34DC1     L1E118  JMP   :C14D      Popall, ret
071
072                 * Calculate P(i+1):
073
074 E63D 21E700     L1E119  LXI   H,:00E7
075 E640 E5                 PUSH  H
076 E641 CDDBE9             CALL  :E9DB      Copy P(i) into 00E7-EA
077 E644 21E300             LXI   H,:00E3
078 E647 CDFBE9             CALL  :E9FB      Copy F from 00E3-E6 into
079                                           MACC
080 E64A E1                 POP   H
081 E64B E5                 PUSH  H
082 E64C CD20EA             CALL  :EA20      Calc F/P(i)
083 E64F E1                 POP   H
084 E650 CD72EA             CALL  :EA72      Calc P(i)+F/P(i)
085 E653 3D                 DCR   A          exp minus 1: divide by 2
086 E654 E67F               ANI   :7F        Skip sign bit
087 E656 C9                 RET
088
089                 * CONSTANTS FOR 'XSQRT':
090
091 E657 7F         L1E275  DATA  :7F        a1: 0.578125
092 E658 D2                 DATA  :D2
093 E659 D0                 DATA  :D0
094 E65A 1C                 DATA  :1C
095                 *
096 E65B 00                 DATA  :00        b1: 0.421875
097 E65C 99                 DATA  :99
098 E65D EE                 DATA  :EE
099 E65E 14                 DATA  :14
100                 *
101 E65F 00         L1E277  DATA  :00        a2: 0.411744
102 E660 94                 DATA  :94
103 E661 00                 DATA  :00
104 E662 00                 DATA  :00
105                 *
106 E663 7F                 DATA  :7F        b2: 0.601289
107 E664 D8                 DATA  :D8
108 E665 00                 DATA  :00
109 E666 00                 DATA  :00
110                 *
111                 ***********
112                 * FPT EXP *
113                 ***********
114                 *
115                 * MACC = E ^ MACC.
116                 *
117                 * Method: Polynomial approximation.
118                 *
119                 * Let E^X = 2^n * 2^d * 2^z:
120                 *     Then X/ln2 = n + d + z.
121                 *        n: integral portion of the real number.
122                 *        d: a discrete fraction (1/8, 3/8, 5/8
123                 *           or 7/8) of the fractional part.
124                 *        z: remainder: -1/8 <= z <= 1/8.
                      Approximation for 2^z:
```

```
126                     *     2^z = a0 + a1*z + a2*z^2 + .... + a5*z^5.
127                     *
128 E667 F5             XEXP    PUSH  PSW
129 E668 C5                     PUSH  B
130 E669 D5                     PUSH  D
131 E66A E5                     PUSH  H
132 E66B 3AD500                 LDA   :00D5     Get exp.byte
133 E66E 32EF00                 STA   :00EF     Save it
134 E671 CDEEE9                 CALL  :E9EE     MACC= ABS(MACC)
135 E674 212BE7                 LXI   H,:E72B   Addr 1/ln2
136 E677 CD59EA                 CALL  :EA59     Calc X/ln2
137 E67A CD1EC2                 CALL  :C21E     Result (n+d+z) on stack
138 E67D CD14E4                 CALL  :E414     Convert MACC to INT (n)
139 E680 CD33E1                 CALL  :E133     n in ABCD
140 E683 CD34C2                 CALL  :C234     Get (n+d+z) from stack
141 E686 B0                     ORA   B
142 E687 B1                     ORA   C
143 E688 CA94E6                 JZ    :E694     Jump if n <= 255
144
145                     * If X too big:
146
147 E68B 3AEF00                 LDA   :00EF     Get exp.byte
148 E68E 2F                     CMA             Take complement
149 E68F B7                     ORA   A         Set flags for error
150 E690 37                     STC             Init error exit
151 E691 C3F5E6                 JMP   :E6F5     Run error, abort
152
153                     * Find d:
154
155 E694 D5             L1E121  PUSH  D         Save n
156 E695 CD54E1                 CALL  :E154     MACC = FRAC (MACC)
157 E698 11FBE6                 LXI   D,:E6FB   Addr FPT(1/8)
158 E69B 2AD500                 LHLD  :00D5
159 E69E B5                     ORA   L
160 E69F CAF9EF                 JZ    :EFF9
161 E6A2 FE7F                   CPI   :7F
162 E6A4 DAB8E6                 JC    :E6B8
163 E6A7 11FFE6                 LXI   D,:E6FF   Addr FPT(3/8)
164 E6AA C3B8E6                 JMP   :E6B8
165 E6AD 07             L1E122  RLC
166 E6AE 07                     RLC
167 E6AF 1103E7                 LXI   D,:E703   Addr FPT(5/8)
168 E6B2 D2B8E6                 JNC   :E6B8
169 E6B5 1107E7                 LXI   D,:E707   Addr FPT(7/8)
170                     *
171 E6B8 EB             L1E123  XCHG            Addr d in HL
172 E6B9 E5                     PUSH  H         Save it
173 E6BA CD6DEA                 CALL  :EA6D     MACC= MACC-d (z)
174 E6BD 5F                     MOV   E,A       Exp. z in E
175 E6BE 3AEF00                 LDA   :00EF     Get exp. X
176 E6C1 07                     RLC             Sign into carry
177 E6C2 F5                     PUSH  PSW       Save sign
178 E6C3 7B                     MOV   A,E       Get exp. z
179 E6C4 DCE4E9                 CC    :E9E4     Evt. change sign
180 E6C7 21E300                 LXI   H,:00E3
181 E6CA CDDBE9                 CALL  :E9DB     Copy z into 00E3-E6
182 E6CD CDDBE9                 CALL  :E9DB     and in 00E7-EA
183 E6D0 2162C4                 LXI   H,:C462   Addr a0 (FPT(1))
184 E6D3 CDFBE9                 CALL  :E9FB     Copy a0 into MACC
185 E6D6 212FE7                 LXI   H,:E72F   Addr table a1-a5
186 E6D9 CDAAE5                 CALL  :E5AA     Calc Taylor sum 2^z
187 E6DC F1                     POP   PSW       Get exp.byte X SHL 1.
```

```
188                                                  sign in CY
189 E6DD D1                    POP   D               Get addr FPT(n/8)
190 E6DE F5                    PUSH  PSW
191 E6DF 211000                LXI   H,:0010         Init offset for table L1E283
192 E6E2 D2E6E6                JNC   :E6E6           Jump if X was positive
193 E6E5 29                    DAD   H               Offset is #0020 for neg.nr.
194 E6E6 19          L1E124    DAD   D               Calc addr in L1E283
195 E6E7 CD59EA                CALL  :EA59           Calc 2^z * 2^d
196 E6EA F1                    POP   PSW             Get CY on sign of X
197 E6EB E1                    POP   H               Get n in H
198 E6EC 7C                    MOV   A,H
199 E6ED D2F2E6                JNC   :E6F2           Jump if X was positive
200 E6F0 2F                    CMA                   ) Else: complement n
201 E6F1 3C                    INR   A               )
202 E6F2 CDB7C1      L1E125    CALL  :C1B7           Add exponents (n+d+z)
203 E6F5 DC4BEA      L1E126    CC    :EA4B           Evt error handling
204 E6F8 C34DC1      L1E127    JMP   :C14D           Popall, ret
205
206                  * CONSTANTS FOR 'XEXP':
207
208 E6FB 7E           L1E279    DATA  :7E             FPT(1/8)
209 E6FC 80                     DATA  :80
210 E6FD 00                     DATA  :00
211 E6FE 00                     DATA  :00
212                  *
213 E6FF 7F           L1E280    DATA  :7F             FPT(3/8)
214 E700 C0                     DATA  :C0
215 E701 00                     DATA  :00
216 E702 00                     DATA  :00
217                  *
218 E703 00           L1E281    DATA  :00             FPT(5/8)
219 E704 A0                     DATA  :A0
220 E705 00                     DATA  :00
221 E706 00                     DATA  :00
222                  *
223 E707 00           L1E282    DATA  :00             FPT(7/8)
224 E708 E0                     DATA  :E0
225 E709 00                     DATA  :00
226 E70A 00                     DATA  :00
227                  *
228                  *
229 E70B 01           L1E283    DATA  :01             2^(1/8)
230 E70C 8B                     DATA  :8B
231 E70D 95                     DATA  :95
232 E70E C2                     DATA  :C2
233                  *
234 E70F 01                     DATA  :01             2^(3/8)
235 E710 A5                     DATA  :A5
236 E711 FE                     DATA  :FE
237 E712 D7                     DATA  :D7
238                  *
239 E713 01                     DATA  :01             2^(5/8)
240 E714 C5                     DATA  :C5
241 E715 67                     DATA  :67
242 E716 2A                     DATA  :2A
243                  *
244 E717 01                     DATA  :01             2^(7/8)
245 E718 EA                     DATA  :EA
246 E719 C0                     DATA  :C0
247 E71A C7                     DATA  :C7
248                  *
249 E71B 00           L1E287    DATA  :00             2^(-1/8)
```

```
250 E71C EA                     DATA  :EA
251 E71D C0                     DATA  :C0
252 E71E C7                     DATA  :C7
253                     *
254 E71F 00                     DATA  :00       2^(-3/8)
255 E720 C5                     DATA  :C5
256 E721 67                     DATA  :67
257 E722 2A                     DATA  :2A
258                     *
259 E723 00                     DATA  :00       2^(-5/8)
260 E724 A5                     DATA  :A5
261 E725 FE                     DATA  :FE
262 E726 D7                     DATA  :D7
263                     *
264 E727 00                     DATA  :00       2^(-7/8)
265 E728 8B                     DATA  :8B
266 E729 95                     DATA  :95
267 E72A C2                     DATA  :C2
268                     *
269                     *
270 E72B 01             L1E291  DATA  :01       1/LN2
271 E72C B8                     DATA  :B8
272 E72D AA                     DATA  :AA
273 E72E 3B                     DATA  :3B
274                     *
275                     *
276 E72F 00             L1E292  DATA  :00       a1: LN2
277 E730 B1                     DATA  :B1       0.69314718057
278 E731 72                     DATA  :72
279 E732 18                     DATA  :18
280                     *
281 E733 7E                     DATA  :7E       a2: ((LN2)^2)/2!
282 E734 F5                     DATA  :F5       0.24022648580
283 E735 FD                     DATA  :FD
284 E736 EF                     DATA  :EF
285                     *
286 E737 7C                     DATA  :7C       a3: ((LN2)^3)/3!
287 E738 E3                     DATA  :E3       0.055504105406
288 E739 58                     DATA  :58
289 E73A 46                     DATA  :46
290                     *
291 E73B 7A                     DATA  :7A       a4: ((LN2)^4)/4!
292 E73C 9D                     DATA  :9D       0.009621738974?
293 E73D A4                     DATA  :A4
294 E73E 81                     DATA  :81
295                     *
296 E73F 77                     DATA  :77       a5: ((LN2)^5)/5!
297 E740 AE                     DATA  :AE       0.001333772937?
298 E741 D1                     DATA  :D1
299 E742 FE                     DATA  :FE
300                     *
301 E743 00                     DATA  :00       End of table
302 E744 00                     DATA  :00
303                     *
304                     *******
305                     * LOG *
306                     *******
307                     *
308                     * MACC = LN (MACC).
309                     *
310                     * Method: Polynomial approximation.
311                     *
```

```
                        * Write X = 2^K * F (normalized written), with
                        * 0.5<= F <1.
                        *   If F < SQR(2)/2: J=K-1, G=2*F.
                        *   If F > SQR(2)/2: J=K,   G=F.
                        * Now X = 2^J * G.
                        *
                        * Assume G=(1+v)/(1-v), then:
                        *       ln(X) = J*ln(2) + ln((1+v)/(1-v)).
                        *
                        * ln((1+v)/(1-v))=2(v+v^3/3+v^5/5+...+v^9/9).
                        * Only terms up to v^9 are used. The term constants
                        * are adjusted for minimum error.
                        *
                        * Exit: 00E3-E6: Last significant summand.
                        *       00E7-EA: v^2.
                        *       00EB-EE: Entry MACC (X).
                        *       MACC:    Result.
                        *       All registers preserved.
                        *
E745 F5         XLN     PUSH  PSW
E746 C5                 PUSH  B
E747 D5                 PUSH  D
E748 E5                 PUSH  H
E749 CDF1EB             CALL  :EBF1     Check contents MACC
E74C CAD0E9             JZ    :E9D0     Run argument error if
                                        MACC = 0
E74F B7                 ORA   A
E750 FAD0E9             JM    :E9D0     Error if nr is negative
E753 CDE9C1             CALL  :C1E9     Sign extend exp (=K)
E756 F5                 PUSH  PSW       Save sign extended exp
E757 3600               MVI   M,:00     Frig exponent
E759 3AD600             LDA   :00D6     Get hibyte mantissa
E75C FEB5               CPI   :B5       Compare with SQR(2)/2
E75E D26AE7             JNC   :E76A     if F < SQR(2)/2
                        *
                        * If F > SQR(2)/2:
                        *
E761 2166C4             LXI   H,:C466   Addr FPT(2)
E764 CD59EA             CALL  :EA59     Calc MACC = 2*F (=G)
E767 F1                 POP   PSW       Get K
E768 3D                 DCR   A         J=K-1
E769 F5                 PUSH  PSW       Save J
                        *
E76A 2162C4     FLNA    LXI   H,:C462   Addr FPT(1)
E76D CD72EA             CALL  :EA72     MACC = G+1
E770 CD1EC2             CALL  :C21E     save G+1 on stack
E773 2166C4             LXI   H,:C466   Addr FPT(2)
E776 CD6DEA             CALL  :EA6D     MACC = G-1
E779 210000             LXI   H,:0000
E77C 39                 DAD   SP        HL=SP
E77D CD20EA             CALL  :EA20     MACC = (G-1)/(G+1) (=v)
E780 33                 INX   SP        )
E781 33                 INX   SP        ) Suppress 4 bytes
E782 33                 INX   SP        ) on stack
E783 33                 INX   SP        )
E784 21E300             LXI   H,:00E3
E787 CDDBE9             CALL  :E9DB     Copy v into 00E3-E6
E78A E5                 PUSH  H         Pnts to 00E7
E78B CD1EC2             CALL  :C21E     Save v on stack
E78E 210000             LXI   H,:0000
E791 39         L1E273  DAD   SP        HL=SP
E792 CD59EA             CALL  :EA59     MACC = v^2
```

```
374 E795 33                 INX   SP        )
375 E796 33                 INX   SP        ) Suppress 4 bytes
376 E797 33                 INX   SP        ) on stack
377 E798 33                 INX   SP        )
378 E799 E1                 POP   H         HL=00E7
379 E79A CDDBE9             CALL  :E9DB     Copy v^2 into 00E7-EA
380 E79D D1                 POP   D         Get J in D
381 E79E 7A                 MOV   A,D       )
382 E79F 17                 RAL             )
383 E7A0 9F                 SBB   A         ) Convert J from 1 byte
384 E7A1 47                 MOV   B,A       ) into 4 byte into ABCD
385 E7A2 4F                 MOV   C,A       )
386 E7A3 CD26E1             CALL  :E126     Copy ABCD into MACC
387 E7A6 CDDEE3             CALL  :E3DE     MACC = INT(MACC)
388 E7A9 21B8E7             LXI   H,:E7B8   Addr ln(2)
389 E7AC CD59EA             CALL  :EA59     MACC=MACC*ln(2) (=J*ln(2))
390 E7AF 21BCE7             LXI   H,:E7BC   Addr Taylor sum constants
391 E7B2 CDAAE5             CALL  :E5AA     Calc Taylor sum (= ln(X)
392 E7B5 C34DC1             JMP   :C14D     Popall, ret
393
394                 * CONSTANTS FOR 'XLN':
395
396 E7B8 00        L1E298   DATA  :00       LN(2)
397 E7B9 B1                 DATA  :B1
398 E7BA 72                 DATA  :72
399 E7BB 18                 DATA  :18
400                 *
401 E7BC 02        L1E299   DATA  :02       b1: FPT (2)
402 E7BD 80                 DATA  :80
403 E7BE 00                 DATA  :00
404 E7BF 00                 DATA  :00
405                 *
406 E7C0 00                 DATA  :00       b3: about 2/3
407 E7C1 AA                 DATA  :AA       0.666666564181
408 E7C2 AA                 DATA  :AA
409 E7C3 A9                 DATA  :A9
410                 *
411 E7C4 7F                 DATA  :7F       b5: about 2/5
412 E7C5 CC                 DATA  :CC       0.400018840613
413 E7C6 CF                 DATA  :CF
414 E7C7 45                 DATA  :45
415                 *
416 E7C8 7F                 DATA  :7F       b7: about 2/7
417 E7C9 91                 DATA  :91       0.2845357266
418 E7CA AE                 DATA  :AE
419 E7CB AB                 DATA  :AB
420                 *
421 E7CC 7E                 DATA  :7E       b9: about 2/9
422 E7CD 80                 DATA  :80       0.125
423 E7CE 00                 DATA  :00
424 E7CF 00                 DATA  :00
425                 *
426 E7D0 00                 DATA  :00       End of table
427 E7D1 00                 DATA  :00
428                 *
429                 *
430                 *
431 E7D2                    END

*****************************
* S Y M B O L   T A B L E *
*****************************
```

FLNA	E76A	L1E117	E618	L1E118	E63A	L1E119	E63D
L1E121	E694	L1E122	E6AD	L1E123	E6B8	L1E124	E6E6
L1E125	E6F2	L1E126	E6F5	L1E127	E6F8	L1E273	E791
L1E275	E657	L1E277	E65F	L1E279	E6FB	L1E280	E6FF
L1E281	E703	L1E282	E707	L1E283	E70B	L1E287	E71B
L1E291	E72B	L1E292	E72F	L1E298	E7B8	L1E299	E7BC
XEXP	E667	XLN	E745	XSQRT	E5F8		

```
002                             ORG   :E7D2
003                     *
004                     *
005                     *
006                     *******
007                     * SIN *
008                     *******
009                     *
010                     * MACC = SIN (MACC)  (Angle expressed in radians).
011                     *
012                     * See XCOS for explanation.
013                     *
014 E7D2 F5             XSIN    PUSH  PSW
015 E7D3 C5                     PUSH  B
016 E7D4 D5                     PUSH  D
017 E7D5 E5                     PUSH  H
018 E7D6 C3E3E7                 JMP   :E7E3     To common part XSIN/XCOS
019                     *
020                     *******
021                     * COS *
022                     *******
023                     *
024                     * MACC = COS (MACC)  (Angle expressed in radians).
025                     *
026                     * Method: Polynomial approximation.
027                     *
028                     * Cos(X) is converted: cos(X) = sin(X+PI/2).
029                     *
030                     * Given X, N and Y are defined for:
031                     *       X/(2*PI) = N + Y; N is integer part.
032                     *
033                     * All arguments are converted to a range -PI/2 to
034                     * +PI/2:
035                     *       sin(N*2*PI+K) = sin(K)
036                     *       sin(PI/2+K)   = sin(PI/2-K)
037                     *       sin(PI*3/2+K) = sin(PI*3/2-K)
038                     *       sin(-PI/2+K)  = sin(-PI/2-K).
039                     *
040                     * Polynomial approx. F(Y) for sin(2*PI*Y) is:
041                     *       F(Y) = a1*Y + a2*Y^3 + ... + a5*Y^9.
042                     *
043 E7D9 F5             XCOS    PUSH  PSW
044 E7DA C5                     PUSH  B
045 E7DB D5                     PUSH  D
046 E7DC E5                     PUSH  H
047 E7DD 2133E8                 LXI   H,:E833   Addr PI/2
048 E7E0 CD72EA                 CALL  :EA72     X = X + PI/2
049
050                     * Entry from XSIN:
051
052 E7E3 213FE8         L1E132  LXI   H,:E83F   Addr PI*2
053 E7E6 CD20EA                 CALL  :EA20     MACC = X/(2*PI) = N+Y
054 E7E9 CD54E1                 CALL  :E154     Get FRAC(MACC) = Y
055 E7EC 21D500                 LXI   H,:00D5   Addr MACC
056 E7EF 7E                     MOV   A,M       Get exp.byte
057 E7F0 E67F                   ANI   :7F       Exp only
058 E7F2 CAFAE7                 JZ    :E7FA     Jump if exp is 0
059 E7F5 FE7E                   CPI   :7E
060 E7F7 DA18E8                 JC    :E818     Jump if exp < 7E
061 E7FA BE             L1E133  CMP   M         Comp masked/non-masked exp
062 E7FB 2162C4                 LXI   H,:C462   Addr FPT (1)
063 E7FE C472EA                 CNZ   :EA72     Add 1 to Y if X negative
```

```
064 E801 2137E8          LXI   H,:E837   Addr FPT (0.25)
065 E804 E5              PUSH  H         Save pntr
066 E805 CD6DEA          CALL  :EA6D     MACC = MACC - 0.25
067 E808 CDEEE9          CALL  :E9EE     Take abs. value
068 E80B 213BE8          LXI   H,:E83B   Addr FPT (0.5)
069 E80E CD6DEA          CALL  :EA6D     MACC = MACC - 0.5
070 E811 CDEEE9          CALL  :E9EE     Take abs. value
071 E814 E1              POP   H         Get addr FPT (0.25)
072 E815 CD6DEA          CALL  :EA6D     MACC = MACC - 0.25
073 E818 21E300  L1E134  LXI   H,:00E3
074 E81B E5              PUSH  H
075 E81C CDD6E9          CALL  :E9D6     Copy MACC into 00E3-E6
076 E81F E3              XTHL            HL=00E3; stack: 00E7
077 E820 CD59EA          CALL  :EA59     MACC = 2 * MACC
078 E823 E1              POP   H         HL=00E7
079 E824 CDDBE9          CALL  :E9DB     Copy 2*MACC into 00E7-EA
080 E827 CD16EA          CALL  :EA16     Clear MACC + reg ABCD
081 E82A 213FE8          LXI   H,:E83F   Addr Taylor sum constants
082 E82D CDAAE5          CALL  :E5AA     Calc Taylor sum
083 E830 C34DC1          JMP   :C14D     Popall, ret
084
085                  * CONSTANTS FOR 'XSIN' AND 'XCOS':
086
087 E833 01       FPHPI  DATA  :01       FPT (PI/2)
088 E834 C9             DATA  :C9
089 E835 0F             DATA  :0F
090 E836 DB             DATA  :DB
091                  *
092 E837 7F       L1E304 DATA  :7F       FPT (0.25)
093 E838 80             DATA  :80
094 E839 00             DATA  :00
095 E83A 00             DATA  :00
096                  *
097 E83B 00       L1E305 DATA  :00       FPT (0.5)
098 E83C 80             DATA  :80
099 E83D 00             DATA  :00
100 E83E 00             DATA  :00
101                  *
102 E83F 03       L1E306 DATA  :03       a1: about PI*2
103 E840 C9             DATA  :C9       6.2831853
104 E841 0F             DATA  :0F
105 E842 DB             DATA  :DB
106                  *
107 E843 86             DATA  :86       a2: about -(PI*2)^3/3!
108 E844 A5             DATA  :A5       -41.341681
109 E845 5D             DATA  :5D
110 E846 E2             DATA  :E2
111                  *
112 E847 07             DATA  :07       a3: about (PI*2)^5/5!
113 E848 A3             DATA  :A3       81.602481
114 E849 34             DATA  :34
115 E84A 78             DATA  :78
116                  *
117 E84B 87             DATA  :87       a4: about -(PI*2)^7/7!
118 E84C 99             DATA  :99       -76.581285
119 E84D 29             DATA  :29
120 E84E 9E             DATA  :9E
121                  *
122 E84F 06             DATA  :06       a5: about (PI*2)^9/9!
123 E850 9F             DATA  :9F       39.760722
124 E851 0A             DATA  :0A
                        DATA  :FB
```

```
126                     *
127 E853 00             DATA  :00       End of table
128 E854 00             DATA  :00
129                     *
130                     *********
131                     * POWER *
132                     *********
133                     *
134                     * MACC = MACC ^ MEM.
135                     *
136                     * Entry: HL points to power in memory.
137                     * Exit:  All registers preserved.
138                     *
139                     * Conditions for a^X:
140                     *       a > 0.
141                     *       ABS (x*ln(a)) in valid range.
142                     *
143                     * Method: a^X = e^(X*ln(a)).
144                     *
145 E855 F5          XPWR    PUSH  PSW
146 E856 C5                  PUSH  B
147 E857 D5                  PUSH  D
148 E858 E5                  PUSH  H
149 E859 E5                  PUSH  H         Save addr X
150 E85A CDF8E9              CALL  :E9F8     Get a in reg ABCD
151 E85D E1                  POP   H         Restore addr X
152 E85E CA6DE8              JZ    :E86D     Abort if a = 0
153 E861 FAD0E9              JM    :E9D0     Argument error if nr < 0
154 E864 CD45E7              CALL  :E745     MACC = ln(a)
155 E867 CD59EA              CALL  :EA59     MACC = X*ln(a)
156 E86A CD67E6              CALL  :E667     MACC = e^(X*ln(a))
157 E86D C34DC1          XPW10   JMP   :C14D     Popall, ret
158                     *
159                     ********
160                     * LOGT *
161                     ********
162                     *
163                     * MACC = LOG (MACC).
164                     *
165                     * Method: log(X) = ln(x) / ln(10).
166                     *
167                     * Exit: All registers preserved.
168                     *
169 E870 F5          XLOG    PUSH  PSW
170 E871 C5                  PUSH  B
171 E872 D5                  PUSH  D
172 E873 E5                  PUSH  H
173 E874 CD45E7              CALL  :E745     MACC = ln(ABS(X))
174 E877 2190E8              LXI   H,:E890   Addr 1/ln(10)
175 E87A CD59EA              CALL  :EA59     MACC = ln(x)/ln(10)
176 E87D C34DC1              JMP   :C14D     Popall, ret
177                     *
178                     ********
179                     * ALOG *
180                     ********
181                     *
182                     * MACC = ALOG (MACC).
183                     *
184                     * Method: 10^X = e^(X*ln(10)).
185                     *
186                     * Exit: All registers preserved.
187                     *
```

```
188 E880 F5          XALOG   PUSH  PSW
189 E881 C5                  PUSH  B
190 E882 D5                  PUSH  D
191 E883 E5                  PUSH  H
192 E884 2190E8              LXI   H,:E890    Addr 1/ln(10)
193 E887 CD20EA              CALL  :EA20      MACC = X*ln(10)
194 E88A CD67E6              CALL  :E667      MACC = e^(X*ln(10))
195 E88D C34DC1              JMP   :C14D      Popall, ret
196
197                  * CONSTANT FOR 'XLOG' AND 'XALOG':
198
199 E890 7F            FLGTI   DATA  :7F        1/ln(10)
200 E891 DE                  DATA  :DE
201 E892 5B                  DATA  :5B
202 E893 D9                  DATA  :D9
203                  *
204                  *******
205                  * TAN *
206                  *******
207                  *
208                  * MACC = TAN (MACC)  (Angle in radians).
209                  *
210                  * Method: tan(X) = sin(X)/cos(X).
211                  *         In-accurate for X close to 0 or close
212                  *         to n*PI/2.
213                  *
214                  * Exit: All registers preserved.
215                  *
216 E894 E5          XTAN    PUSH  H
217 E895 CD1EC2              CALL  :C21E      Save X on stack
218 E898 CDD9E7              CALL  :E7D9      MACC = cos(X)
219 E89B 21EF00              LXI   H,:00EF
220 E89E CD1CE1              CALL  :E11C      Store cos(X) in 00EF-F2
221 E8A1 CD34C2              CALL  :C234      Get X from stack
222 E8A4 CDD2E7              CALL  :E7D2      MACC = sin(X)
223 E8A7 CD08E1              CALL  :E108      MACC = sin(X)/cos(X)
224 E8AA E1                  POP   H
225 E8AB C9                  RET
226                  *
227                  ********
228                  * ATAN *
229                  ********
230                  *
231                  * MACC = ATAN (MACC)  (Angle expressed in radians).
232                  *
233                  * Method: Polynomial approximation.
234                  *
235                  * ATAN(Z) for -0.25 <= Z <= 0.25 approximated by:
236                  *       F(X) = X*(1 - Q1*X^2 + Q2*X^4 - Q3*X^6).
237                  *
238                  * To cope with range:
239                  *       ATAN(-Z) = - ATAN(Z).
240                  *       ATAN(Z) = a(k) + ATAN((Z-b(k))/(Z*b(k)+1)),
241                  *                 with k = 1, 2 or 3,
242                  *                 a(k) = k*PI/7,
243                  *                 b(k) = TAN(a(k))
244                  *
245                  * Values for k:
246                  *       k=0 if ABS(Z) < 0.25
247                  *       k=1 if 0.25 < ABS(Z) < 0.75
248                  *       k=2 if 0.75 < ABS(Z) < 2
249                  *       k=3 if ABS(Z) > 2.
```

```
250                     *
251                     * Then X = (Z-b(k))/(Z*b(k)+1), and
252                     *      ATAN(Z) =  a(k) + F(X), if Z >= 0
253                     *      ATAN(Z) = -a(k) - F(X), if Z < 0.
254                     *
255 E8AC F5          XATAN   PUSH  PSW
256 E8AD C5                  PUSH  B
257 E8AE D5                  PUSH  D
258 E8AF E5                  PUSH  H
259 E8B0 CD71EB              CALL  :EB71     Check if Z=0
260 E8B3 CA43E9              JZ    :E943     Then abort
261 E8B6 F5                  PUSH  PSW       Save exp byte
262 E8B7 CDEEE9              CALL  :E9EE     reg ABCD = ABS(Z)
263 E8BA 21EF00              LXI   H,:00EF
264 E8BD CDDBE9              CALL  :E9DB     Copy ABS(Z) into 00EF-F2
265
266                     * Calculate k:
267
268 E8C0 FE40                CPI   :40
269 E8C2 DAD3E8              JC    :E8D3     Jump if exp < #40
270 E8C5 FE7F                CPI   :7F
271 E8C7 3E01                MVI   A,:01
272 E8C9 CAE6E8              JZ    :E8E6     k=1 if exp=#7F
273 E8CC 215EC4              LXI   H,:C45E   Addr FPT(0)
274 E8CF E5                  PUSH  H
275 E8D0 C315E9              JMP   :E915     Cont with k=1, a(k)=0
276 E8D3 FE01        L1E141  CPI   :01
277 E8D5 3E02                MVI   A,:02
278 E8D7 CAE6E8              JZ    :E8E6     k=2 if exp=1
279 E8DA D2E3E8              JNC   :E8E3     k=3 if exp >1
280 E8DD 78                  MOV   A,B       Get hibyte mantissa
281 E8DE 07                  RLC
282 E8DF 07                  RLC
283 E8E0 3E01                MVI   A,:01     k=1 if (B)= 10...
284                                          k=2 if (B)= 11...
285 E8E2 3F                  CMC
286 E8E3 3F          L1E142  CMC
287 E8E4 CE00                ACI   :00
288                     *
289 E8E6 87          L1E143  ADD   A         Final k in A
290 E8E7 87                  ADD   A
291 E8E8 87                  ADD   A         *8
292 E8E9 213EE9              LXI   H,:E93E   Startaddr for a,b table
293 E8EC 5F                  MOV   E,A       )
294 E8ED 1600                MVI   D,:00     ) offset in DE
295 E8EF 19                  DAD   D
296 E8F0 E5                  PUSH  H         Addr a(k)
297 E8F1 110400              LXI   D,:0004
298 E8F4 19                  DAD   D
299 E8F5 E5                  PUSH  H         Addr b(k)
300 E8F6 CD59EA              CALL  :EA59     MACC = Z*b(k)
301 E8F9 2162C4              LXI   H,:C462   Addr FPT(1)
302 E8FC CD72EA              CALL  :EA72     MACC = Z*b(k)+1
303 E8FF 21DF00              LXI   H,:00DF
304 E902 CDDBE9              CALL  :E9DB     (Z*b(k)+1) into 00DF-E2
305 E905 21EF00              LXI   H,:00EF
306 E908 CDFBE9              CALL  :E9FB     ABS(Z) in MACC
307 E90B E1                  POP   H         Addr b(k)
308 E90C CD6DEA              CALL  :EA6D     MACC = Z-b(k)
309 E90F 21DF00              LXI   H,:00DF
310 E912 CD20EA              CALL  :EA20     MACC = X =
311                                          = (Z-b(k))/(Z*b(k)+1)
```

```
312 E915 21EF00      L1E144  LXI   H,:00EF
313 E918 E5                  PUSH  H
314 E919 E5                  PUSH  H
315 E91A CDD6E9              CALL  :E9D6     Copy X into 00EF-F2
316 E91D E1                  POP   H
317 E91E CD59EA              CALL  :EA59     MACC = X^2
318 E921 21E300              LXI   H,:00E3
319 E924 CDDBE9              CALL  :E9DB     Copy X^2 into 00E3-E6
320 E927 CDDBE9              CALL  :E9DB     Copy X^2 into 00E7-EA
321 E92A 2162C4              LXI   H,:C462   Addr FPT(1)
322 E92D CDFBE9              CALL  :E9FB     Copy FPT(1) into MACC
323 E930 215EE9              LXI   H,:E95E   Start table Taylor constants
324 E933 CDAAE5              CALL  :E5AA     Calc Taylor sum
325 E936 E1                  POP   H
326 E937 CD59EA              CALL  :EA59     Taylor sum * X  (=F(X))
327 E93A E1                  POP   H
328 E93B CD72EA              CALL  :EA72     Add a(k) (= ATAN(Z))
329 E93E F1                  POP   PSW       Get orig. exp byte
330 E93F B7                  ORA   A         Was Z negative ?
331 E940 FCE4E9              CM    :E9E4     Then MACC = - ATAN(Z)
332 E943 C34DC1      L1E145  JMP   :C14D     Popall, ret
333
334                  * CONSTANTS FOR 'XATN':
335
336 E946 7F          FATC1   DATA  :7F       a(1): PI/7
337 E947 E5                  DATA  :E5       0.4487989506
338 E948 C8                  DATA  :C8
339 E949 FA                  DATA  :FA
340                  *
341 E94A 7F                  DATA  :7F       b(1): TAN(a(1))
342 E94B F6                  DATA  :F6       0.4815746188
343 E94C 90                  DATA  :90
344 E94D F3                  DATA  :F3
345                  *
346 E94E 00                  DATA  :00       a(2): 2*PI/7
347 E94F E5                  DATA  :E5       0.8975979011
348 E950 C8                  DATA  :C8
349 E951 FA                  DATA  :FA
350                  *
351 E952 01                  DATA  :01       b(2): TAN(a(1))
352 E953 A0                  DATA  :A0       1.253960337
353 E954 81                  DATA  :81
354 E955 C6                  DATA  :C6
355                  *
356 E956 01                  DATA  :01       a(3): 3*PI/7
357 E957 AC                  DATA  :AC       1.346396852
358 E958 56                  DATA  :56
359 E959 BB                  DATA  :BB
360                  *
361 E95A 03                  DATA  :03       b(3): TAN(a(3))
362 E95B 8C                  DATA  :8C       4.381286272
363 E95C 33                  DATA  :33
364 E95D 7F                  DATA  :7F
365                  *
366 E95E FF          FATPL   DATA  :FF       Q1: about -1/3
367 E95F AA                  DATA  :AA       -0.333329573
368 E960 AA                  DATA  :AA
369 E961 2D                  DATA  :2D
370                  *
371 E962 7E                  DATA  :7E       Q2: about 1/5
372 E963 CC                  DATA  :CC       0.199641035
373 E964 6E                  DATA  :6E
```

```
374 E965 B3                     DATA  :B3
375                     *
376 E966 FE                     DATA  :FE       Q3: about -1/7
377 E967 86                     DATA  :86       -0.131779888
378 E968 F1                     DATA  :F1
379 E969 4F                     DATA  :4F
380                     *
381 E96A 00                     DATA  :00       End of table
382 E96B 00                     DATA  :00
383                     *
384                     ********
385                     * ASIN *
386                     ********
387                     *
388                     * MACC = ASIN (MACC).  Result in radians.
389                     *
390                     * Range: -PI/2 < X < PI/2.
391                     *
392                     * Method: ASIN(X) = ATAN(X/SQR(1-x^2)).
393                     *
394                     * Exit: All registers preserved.
395                     *
396 E96C F5          XASIN   PUSH  PSW
397 E96D C5                  PUSH  B
398 E96E D5                  PUSH  D
399 E96F E5                  PUSH  H
400 E970 CDF8E9              CALL  :E9F8     Get X in reg ABCD
401 E973 5F                  MOV   E,A       Exp byte in E
402 E974 E67F                ANI   :7F       Mask sign
403 E976 FE01                CPI   :01
404 E978 DA99E9              JC    :E999     Jump if in range
405 E97B C294E9              JNZ   :E994     If >2 or <1
406 E97E 78                  MOV   A,B       )
407 E97F E67F                ANI   :7F       ) Check if mantissa
408 E981 B1                  ORA   C         ) = 80 00 00 (= +/- 1)
409 E982 B2                  ORA   D         )
410 E983 C2D0E9              JNZ   :E9D0     Error if not
411 E986 7B                  MOV   A,E       Get exp
412 E987 B7                  ORA   A         Set flags on it
413 E988 2133E8              LXI   H,:E833   Addr PI/2
414 E98B CD12E1              CALL  :E112     Copy PI/2 into MACC
415 E98E FCE4E9              CM    :E9E4     If nr <0: MACC = -PI/2
416 E991 C34DC1      FASRET  JMP   :C14D     Popall, ret
417                     *
418 E994 FE40        FAS10   CPI   :40
419 E996 DAD0E9              JC    :E9D0     Error if exp <#40
420 E999 CD1EC2      FAS20   CALL  :C21E     Save X on stack
421 E99C 210000              LXI   H,:0000
422 E99F 39                  DAD   SP        HL=SP
423 E9A0 CD59EA              CALL  :EA59     MACC = X^2
424 E9A3 CDE4E9              CALL  :E9E4     MACC = -X^2
425 E9A6 2162C4              LXI   H,:C462   Addr FPT(1)
426 E9A9 CD72EA              CALL  :EA72     MACC = 1-X^2
427 E9AC CDF8E5              CALL  :E5F8     MACC = SQR(1-X^2)
428 E9AF 21EF00              LXI   H,:00EF
429 E9B2 CD1CE1              CALL  :E11C     SQR(1-X^2) in 00EF-F2
430 E9B5 CD34C2              CALL  :C234     Get X from stack in MACC
431 E9B8 CD08E1              CALL  :E108     MACC = X/(SQR(1-X^2))
432 E9BB CDACE8              CALL  :E8AC     MACC = ATAN (MACC)
433 E9BE C391E9              JMP   :E991     Ready
434                     *
435                     *
```

```
                        ********
                        * ACOS *
                        ********
                        *
                        * MACC = ACOS (MACC). Result in radians.
                        *
                        * Range: 0 < X < PI.
                        *
                        * Method: ACOS(X) = PI/2 - ASIN(X).
                        *
                        * Exit: All registers preserved.
                        *
E9C1 CD6CE9             XACOS   CALL  :E96C     MACC = ASIN(X)
E9C4 CD4AE1                     CALL  :E14A     MACC = -ASIN(X)
E9C7 E5                         PUSH  H
E9C8 2133E8                     LXI   H,:E833   Addr PI/2
E9CB CDAAED                     CALL  :EDAA     MACC = PI/2-ASIN(X)
E9CE E1                         POP   H
E9CF C9                         RET

                        * Error exit:

E9D0 CD5EC0             FASER   CALL  :C05E     Run argument error
E9D3 C391E9                     JMP   :E991     Abort
                        *
                        ***********************************************
                        * COPY MACC INTO OPERAND AND INTO A,B,C,D *
                        ***********************************************
                        *
                        * Entry: HL points to operand.
                        * Exit:  HL points past operand.
                        *        AFBCD set as for ATEST.
                        *
                        * From ASTORE used to store reg A,B,C,D into
                        * an operand, pointed at by HL.
                        *
E9D6 E5                 ASAVE   PUSH  H
E9D7 CDF8E9                     CALL  :E9F8     Copy MEM into MACC and ABCD
E9DA E1                         POP   H
E9DB 77                 ASTORE  MOV   M,A       )
E9DC 23                         INX   H         )
E9DD 70                         MOV   M,B       ) Copy reg A,B,C,D into MEM
E9DE 23                         INX   H         )
E9DF 71                         MOV   M,C       )
E9E0 23                         INX   H         )
E9E1 72                         MOV   M,D       )
E9E2 23                         INX   H
E9E3 C9                         RET
                        *
                        ********************************
                        * SUBROUTINE CHANGE SIGN MACC *
                        ********************************
                        *
E9E4 CDF1EB             ACHGS   CALL  :EBF1     Check if MACC empty
E9E7 C8                         RZ              Then ready
E9E8 0180FF                     LXI   B,:FF80   Set mask
E9EB C3F1E9                     JMP   :E9F1     Change sign bit
                        *
                        ******************************
                        * SUBROUTINE FPT ABS (MACC) *
                        ******************************
                        *
```

```
498                     * From ATEST also used to copy MACC into ABCD.
499                     * From L1E158 used to copy operand (pointed at
500                     * by HL) into ABCD and into MACC.
501                     *
502 E9EE 01007F        L1E155  LXI   B,:7F00   Set mask
503 E9F1 21D500        L1E156  LXI   H,:00D5   Addr MACC
504 E9F4 78                    MOV   A,B       Mask in A
505 E9F5 A6                    ANA   M         AND exp byte with mask
506 E9F6 A9                    XRA   C         Set sign bit = 0
507 E9F7 77                    MOV   M,A       Update exp byte MACC
508                     *
509 E9F8 21D500        ATEST   LXI   H,:00D5   Addr MACC
510 E9FB CDF4EB        L1E158  CALL  :EBF4     Check if MEM = 0, get
511                             exp byte in A
512 E9FE CA16EA                 JZ    :EA16     Then clear MACC + ABCD
513 EA01 5F                     MOV   E,A       exp byte in E
514 EA02 23                     INX   H         )
515 EA03 46                     MOV   B,M       )
516 EA04 23                     INX   H         ) Mantissa from MEM
517 EA05 4E                     MOV   C,M       ) into BCD
518 EA06 23                     INX   H         )
519 EA07 56                     MOV   D,M       )
520 EA08 21D500                 LXI   H,:00D5   Addr MACC
521 EA0B C317EB                 JMP   :EB17     Copy ABCD into MACC;
522                             exp from E in A, flags
523                             set on exp ORI 01
524                     *
525                     *
526                     *
527 EA0E                        END
```

```
****************************
* S Y M B O L   T A B L E *
****************************
```

ACHGS	E9E4	ASAVE	E9D6	ASTORE	E9DB	ATEST	E9F8
FAS10	E994	FAS20	E999	FASER	E9D0	FASRET	E991
FATC1	E946	FATPL	E95E	FLGTI	E890	FPHPI	E833
L1E132	E7E3	L1E133	E7FA	L1E134	E818	L1E141	E8D3
L1E142	E8E3	L1E143	E8E6	L1E144	E915	L1E145	E943
L1E155	E9EE	L1E156	E9F1	L1E158	E9FB	L1E304	E837
L1E305	E83B	L1E306	E83F	XACOS	E9C1	XALOG	E880
XASIN	E96C	XATAN	E8AC	XCOS	E7D9	XLOG	E870
XPW10	E86D	XPWR	E855	XSIN	E7D2	XTAN	E894

```
                        ORG   :EA0E
                *
                *
                *
                ****************************************
                * COPY OPERAND INTO REGISTERS B,C,D,E *
                ****************************************
                *
                * Entry: HL points to operand.
                * Exit:  HL points to last byte of operand.
                *        AF preserved.
                *
EA0E 46         L1E159  MOV   B,M
EA0F 23                 INX   H
EA10 4E                 MOV   C,M
EA11 23                 INX   H
EA12 56                 MOV   D,M
EA13 23                 INX   H
EA14 5E                 MOV   E,M
EA15 C9                 RET
                *
                ************************************
                * CLEAR MACC AND REGISTERS A,B,C,D *
                ************************************
                *
EA16 21D500     AZERO   LXI   H,:00D5   Addr MACC
EA19 AF                 XRA   A         )
EA1A 47                 MOV   B,A       ) Clear ABCD
EA1B 4F                 MOV   C,A       )
EA1C 57                 MOV   D,A       )
EA1D C3DBE9             JMP   :E9DB     Clear MACC
                *
                ***************************
                * FPT DIVIDE SUBROUTINE *
                ***************************
                *
                * MACC = MACC / MEM. Rounded quotient in MACC
                * and registers ABCD, exponent in E.
                *
                * Entry: HL points to operand.
                * Exit:  CY=1: Overflow, result invalid.
                *        CY=0: Result in ABCD, EHL corrupted.
                *
EA20 CDF4EB     ADIV    CALL  :EBF4     Test if MEM=0; exp byte in A
EA23 CA54EA             JZ    :EA54     Then run divide by 0 error
EA26 F5                 PUSH  PSW       Save exp MEM
EA27 E680               ANI   :80       Sign bit only
EA29 47                 MOV   B,A       Preserve sign
EA2A F1                 POP   PSW       Get exp MEM
EA2B E67F               ANI   :7F       Skip sign bit
EA2D 2F                 CMA             ) 2-compl of exponent
EA2E 3C                 INR   A         )
EA2F FEC0               CPI   :C0       Overflow in sign bit ?
EA31 CA46EA             JZ    :EA46     Then run overflow error
EA34 E67F               ANI   :7F       Only compl. exp MEM
EA36 B0                 ORA   B         Add sign
EA37 CD1DEB             CALL  :EB1D     Subtract exponents
EA3A DA4BEA             JC    :EA4B     Evt. run overflow error
EA3D CA16EA             JZ    :EA16     If zero result: clear MACC +
                                        ABCD
EA40 CD4AEC             CALL  :EC4A     Run fixed division
                        JNC   :EB06     Round up if no overflow
```

```
064
065                     * If overflow:
066
067 EA46 CD4BC0      OVERF   CALL  :C04B      Run overflow error
068 EA49 37                 STC               Flag error
069 EA4A C9                 RET
070                     *
071                     ********************
072                     * ERROR HANDLING *
073                     ********************
074                     *
075                     * Entry: S=1: Overflow error.
076                     *         S=0: Underflow error.
077                     *         Z=1: Divide by zero error.
078                     *
079 EA4B FA46EA      OVUNF   JM    :EA46      Evt run overflow error
080 EA4E CD65C0      UNDRF   CALL  :C065      Run underflow error
081 EA51 C316EA             JMP   :EA16      Clear MACC + ABCD
082 EA54 CD6CC0      DIV0    CALL  :C06C      Run divide by 0 error
083 EA57 37                 STC               Flag error
084 EA58 C9                 RET
085                     *
086                     ***********************************
087                     * FPT MULTIPLICATION SUBROUTINE *
088                     ***********************************
089                     *
090                     * MACC = MACC * MEM. Result in MACC and in
091                     * registers A,B,C,D.
092                     *
093                     * Entry: HL points to operand in memory.
094                     * Exit:  CY=1: Overflow; result invalid.
095                     *         CY=0: Result in ABCD. EHL corrupted.
096                     *
097 EA59 CDF4EB      AMUL    CALL  :EBF4      Test if MEM=0; exp byte in A
098 EA5C C41DEB             CNZ   :EB1D      Add exponents if not
099 EA5F DA4BEA             JC    :EA4B      Evt run error
100 EA62 CA16EA             JZ    :EA16      Result 0: Clear MACC + ABCD
101 EA65 CD00EC             CALL  :EC00      Multiply mantissa's
102
103                     * Normalise if necessary:
104
105 EA68 78                 MOV   A,B        1st product
106 EA69 B7                 ORA   A
107 EA6A C300EB             JMP   :EB00      Common exit with MUL/DIV
108                     *
109                     ****************************
110                     * FPT SUBTRACT SUBROUTINE *
111                     ****************************
112                     *
113                     * MACC = MACC - MEM.
114                     *
115                     * Entry: HL points to operand in memory.
116                     * Exit:  CY=1: Overflow.
117                     *         CY=0: Result in ABCD. EHL corrupted.
118                     *
119 EA6D 0680        ASUB    MVI   B,:80      Mask to change sign of
120                                           operand
121 EA6F C374EA             JMP   :EA74      Into AADD
122                     *
123                     ***********************
124                     * FPT ADD SUBROUTINE *
125                     ***********************
```

```
126                     *
127                     * MACC = MACC - MEM.
128                     *
129                     * Entry: HL points to operand in memory.
130                     * Exit:  CY=1: Overflow.
131                     *        CY=0: Result in ABCD. EHL corrupted.
132                     *
133 EA72 0600           AADD    MVI   B,:00     Zero mask
134 EA74 3E7F           AD10    MVI   A,:7F     Most possible value
135 EA76 32DE00                 STA   :00DE     Set MACC >> MEM
136 EA79 CDF4EB                 CALL  :EBF4     Test if MEM =0; exp in A
137 EA7C CAF8E9                 JZ    :E9F8     Then clear MACC + ABCD
138 EA7F 78                     MOV   A,B       Get mask
139 EA80 AE                     XRA   M         XOR with exp (ADD: gives
140                                             exp; SUB: gives -exp)
141 EA81 23                     INX   H
142 EA82 46                     MOV   B,M       )
143 EA83 23                     INX   H         )
144 EA84 4E                     MOV   C,M       ) Copy mantissa MEM into B
145 EA85 23                     INX   H         )
146 EA86 56                     MOV   D,M       )
147 EA87 5F                     MOV   E,A       Exp in E
148 EA88 21D500                 LXI   H,:00D5   Addr MACC
149 EA8B 7E                     MOV   A,M       Get exp MACC
150 EA8C AB                     XRA   E         XOR with exp MEM
151 EA8D E680                   ANI   :80       Sign only
152 EA8F 32D900                 STA   :00D9     Store #80 if different signs
153 EA92 CDF4EB                 CALL  :EBF4     Test if MACC=0; exp in A
154 EA95 CA11EB                 JZ    :EB11     Jump if true
155 EA98 D5                     PUSH  D
156 EA99 7B                     MOV   A,E       Get exp MEM
157 EA9A CDE9C1                 CALL  :C1E9     Sign extend
158 EA9D 5F                     MOV   E,A       Ext exp MEM in E
159 EA9E 7E                     MOV   A,M       Get exp MACC
160 EA9F CDE9C1                 CALL  :C1E9     Sign extend
161 EAA2 93                     SUB   E         Calc difference
162 EAA3 D1                     POP   D
163 EAA4 32DE00                 STA   :00DE     Save it
164 EAA7 FAB2EA                 JM    :EAB2     If exp MACC < exp MEM:
165                                             exchange ABCD and MACC
166 EAAA FE19                   CPI   :19       Total bits in mantissa
167 EAAC DAC6EA                 JC    :EAC6     OK if difference between
168                                             both nrs <#19 in exp
169 EAAF C3F8E9                 JMP   :E9F8     Else: Result is zero in
170                                             MACC and ABCD
171
172                     * Exchange MACC and ABCD:
173
174 EAB2 FEE7           L1E169  CPI   :E7       Total bits in mantissa
175 EAB4 DA16EB                 JC    :EB16     If difference not too big
176 EAB7 73                     MOV   M,E       Ext exp MEM in MACC
177 EAB8 2F                     CMA             )
178 EAB9 3C                     INR   A         ) A = ext exp old MACC
179 EABA 23                     INX   H
180 EABB 5E                     MOV   E,M       ) Exchange 1st byte MACC
181 EABC 70                     MOV   M,B       ) mantissa and byte in B
182 EABD 43                     MOV   B,E       )
183 EABE 23                     INX   H
184 EABF 5E                     MOV   E,M       ) Exchange 2nd byte MACC
185 EAC0 71                     MOV   M,C       ) mantissa and byte in C
186 EAC1 4B                     MOV   C,E       )
187 EAC2 23                     INX   H
```

```
188 EAC3 5E                 MOV   E,M       ) Exchange 3rd byte MACC
189 EAC4 72                 MOV   M,D       ) mantissa and byte in D
190 EAC5 53                 MOV   D,E       )
191                                         Now orig MACC in ABCD and
192                                         orig MEM in MACC
193 EAC6 1E00       L1E170  MVI   E,:00
194 EACB CD55EB             CALL  :EB55     Shift BCDE right A places
195 EACB 3AD900             LDA   :00D9     Get result XOR sign bits
196 EACE B7                 ORA   A
197 EACF 21D800             LXI   H,:00D8   Addr lobyte MACC
198 EAD2 FAEFEA             JM    :EAEF     Jump if different signbits
199
200                 * If both signs equal:
201
202 EAD5 7E                 MOV   A,M       )
203 EAD6 82                 ADD   D         )
204 EAD7 57                 MOV   D,A       )
205 EAD8 2B                 DCX   H         )
206 EAD9 7E                 MOV   A,M       ) Add mantissa MACC to BCD
207 EADA 89                 ADC   C         ) Result in BCD.
208 EADB 4F                 MOV   C,A       )
209 EADC 2B                 DCX   H         )
210 EADD 7E                 MOV   A,M       )
211 EADE 88                 ADC   B         )
212 EADF 47                 MOV   B,A       )
213 EAE0 D206EB             JNC   :EB06     Jump if no overflow
214 EAE3 CD70EB             CALL  :EB70     Else: shift BCDE right 1 bit
215 EAE6 CDD9EB             CALL  :EBD9     Incr exponent
216 EAE9 DA46EA             JC    :EA46     Evt run overflow error
217 EAEC C306EB             JMP   :EB06     Round up
218
219                 * If both signs not equal:
220
221 EAEF AF           L1E171  XRA   A         )
222 EAF0 93                 SUB   E         ) Compl exp in E
223 EAF1 5F                 MOV   E,A       )
224 EAF2 7E                 MOV   A,M       )
225 EAF3 9A                 SBB   D         )
226 EAF4 57                 MOV   D,A       ) Subtract BCD from mantissa
227 EAF5 2B                 DCX   H         ) MACC. Result in BCD.
228 EAF6 7E                 MOV   A,M       )
229 EAF7 99                 SBB   C         )
230 EAF8 4F                 MOV   C,A       )
231 EAF9 2B                 DCX   H         )
232 EAFA 7E                 MOV   A,M       )
233 EAFB 98                 SBB   B         )
234 EAFC 47                 MOV   B,A       )
235 EAFD DC7DEB             CC    :EB7D     Correct if overflow
236 EB00 F496EB     AD10A   CP    :EB96     Evt normalize BCDE
237 EB03 F216EA             JP    :EA16     and clear MACC + ABCD
238
239                 * Normal exit:
240
241 EB06 CDC3EB       ADD11 CALL  :EBC3     Round up BCD, result in MACC
242                                         exp in E
243 EB09 DA46EA             JC    :EA46     Evt run overflow error
244 EB0C 7B         L1E174  MOV   A,E       Get exponent
245 EB0D F601               ORI   :01       Set flags on exp OR 1
246 EB0F 7B                 MOV   A,E       Exp in A
247 EB10 C9                 RET
248
249                 * If operand = 0:
```

```
250
251 EB11 3E80       L1E175  MVI   A,:80
252 EB13 32DE00             STA   :00DE     (00DE)=#80
253 EB16 7B         L1E176  MOV   A,E       Get exponent
254 EB17 CDDBE9     L1E177  CALL  :E9DB     Copy ABCD into MACC
255 EB1A C30CEB             JMP   :EB0C     Take normal exit
256                 *
257                 ************************
258                 * FPT: ADD EXPONENTS *
259                 ************************
260                 *
261                 * Adds the exponent of the MACC to the exponent
262                 * of a operand in memory.
263                 *
264                 * Entry: HL points to FPT number in memory.
265                 *        A contains its exponent.
266                 *        Other number in MACC.
267                 * Exit:   Z=1:       MACC=0; HL=00D5
268                 *        CY=1:       Overflow: HL=00D5; MACC pres.
269                 *                    A: Sum of signed exponents SHL 1
270                 *        Z=0, CY=0:  O.K.: HL=00D5; sum of exponents
271                 *                    in MACC.
272                 *
273 EB1D 47         MDEX    MOV   B,A       Exp MEM in B
274 EB1E 23                 INX   H
275 EB1F 4E                 MOV   C,M       )
276 EB20 23                 INX   H         ) Copy mantissa MEM in CDE
277 EB21 56                 MOV   D,M       )
278 EB22 23                 INX   H         )
279 EB23 5E                 MOV   E,M       )
280 EB24 CDF1EB             CALL  :EBF1     Test MACC=0; Exp MACC in A
281 EB27 C8                 RZ              Abort if MACC=0, Z=1
282 EB28 78                 MOV   A,B       Get exp MEM in A
283 EB29 CDE9C1             CALL  :C1E9     Sign extend
284 EB2C CDBAC1             CALL  :C1BA     Add exponents, result in MAC
285 EB2F D8                 RC              Abort if overflow, CY=1
286 EB30 78                 MOV   A,B       Get orig exp MEM
287 EB31 E680               ANI   :80       sign bit only
288 EB33 AE                 XRA   M         Evt correct sign
289 EB34 77                 MOV   M,A       Exp back into MACC
290 EB35 3E01               MVI   A,:01
291 EB37 B7                 ORA   A
292 EB38 C9                 RET
293                 *
294                 ************************************
295                 * SHIFT BCDE LEFT (A) POSITIONS *
296                 ************************************
297                 *
298                 * Exit: AF preserved.
299                 *
300 EB39 F5         LSHN    PUSH  PSW
301 EB3A 6F                 MOV   L,A       Nr of shifts in L
302 EB3B 2D         L1E180  DCR   L
303 EB3C FA46EB             JM    :EB46     Abort if ready
304 EB3F B7                 ORA   A         Clear CY
305 EB40 CD48EB             CALL  :EB48     Shift BCDE left 1 position
306 EB43 C33BEB             JMP   :EB3B     Next shift
307 EB46 F1         L1E182  POP   PSW
308 EB47 C9                 RET
309                 *
310                 *
```

```
312                     **********************
313                     * MULTIPLY BCDE * 2 *
314                     **********************
315                     *
316                     * Shifts BCDE left 1 position. Entry CY goes to
317                     * lsb of E.
318                     *
319                     * Exit: A corrupted, HL preserved.
320                     *       F set on result.
321                     *
322 EB48 7B          L1E183  MOV   A,E
323 EB49 17                  RAL                 Shift left E
324 EB4A 5F                  MOV   E,A
325 EB4B 7A                  MOV   A,D
326 EB4C 17                  RAL                 Shift left D
327 EB4D 57                  MOV   D,A
328 EB4E 79                  MOV   A,C
329 EB4F 17                  RAL                 Shift left C
330 EB50 4F                  MOV   C,A
331 EB51 78                  MOV   A,B
332 EB52 8F                  ADC   A             B=2*B+CY
333 EB53 47                  MOV   B,A
334 EB54 C9                  RET
335                     *
336                     ************************************
337                     * SHIFT BCDE RIGHT (A) POSITIONS *
338                     ************************************
339                     *
340 EB55 2E08        RSHN    MVI   L,:0008     Nr of shifts for 1 byte
341 EB57 BD          L1E185  CMP   L
342 EB58 FA64EB              JM    :EB64       Jump if A<8
343
344                     * Shift 8 bits right:
345
346 EB5B 5A                  MOV   E,D         )
347 EB5C 51                  MOV   D,C         ) Shift 8 pos in one
348 EB5D 48                  MOV   C,B         ) time
349 EB5E 0600                MVI   B,:00       )
350 EB60 95                  SUB   L           Update nr of shifts left
351 EB61 C257EB              JNZ   :EB57       Again if not ready
352
353                     * Shift 1 bit:
354
355 EB64 B7          L1E186  ORA   A
356 EB65 C8                  RZ                Abort if ready
357 EB66 6F                  MOV   L,A         L is nr of shifts
358 EB67 B7          L1E187  ORA   A
359 EB68 CD70EB              CALL  :EB70       Shift BCDE right one bit
360 EB6B 2D                  DCR   L           Update shift count
361 EB6C C267EB              JNZ   :EB67       Again if not ready
362 EB6F C9                  RET
363                     *
364                     **********************
365                     * DIVIDE BCDE BY 2 *
366                     **********************
367                     *
368                     * Shifts contents BCDE right 1 position.
369                     *
370                     * Exit: AF corrupted, HL preserved.
371                     *
372 EB70 78          L1E188  MOV   A,B
373 EB71 1F                  RAR               Shift right B
```

```
374 EB72 47                MOV   B,A
375 EB73 79                MOV   A,C
376 EB74 1F                RAR                 Shift right C
377 EB75 4F                MOV   C,A
378 EB76 7A                MOV   A,D
379 EB77 1F                RAR                 Shift right D
380 EB78 57                MOV   D,A
381 EB79 7B                MOV   A,E
382 EB7A 1F                RAR                 Shift right E
383 EB7B 5F                MOV   E,A
384 EB7C C9                RET
385                *
386                ****************************************
387                * CHANGE SIGN OF A NUMBER IN MEMORY *
388                * NEGATE CONTENTS REGISTERS B,C,D,E *
389                ****************************************
390                *
391                * Entry: HL points to 1st byte mantissa.
392                *
393 EB7D 2B        L1E189  DCX   H           Pnts to exp
394 EB7E 7E                MOV   A,M         Get exp
395 EB7F EE80              XRI   :80         Change sign bit
396 EB81 77                MOV   M,A
397 EB82 AF        L1E190  XRA   A
398 EB83 6F                MOV   L,A         L=0
399 EB84 93                SUB   E
400 EB85 5F                MOV   E,A         Negate E
401 EB86 7D                MOV   A,L
402 EB87 9A                SBB   D
403 EB88 57                MOV   D,A         Negate D
404 EB89 7D                MOV   A,L
405 EB8A 99                SBB   C
406 EB8B 4F                MOV   C,A         Negate C
407 EB8C 7D                MOV   A,L
408 EB8D 98                SBB   B
409 EB8E 6F                MOV   L,A         Negated B in L
410 EB8F A0                ANA   B
411 EB90 17                RAL               msb into CY
412 EB91 45                MOV   B,L         B = negated B
413 EB92 7D                MOV   A,L
414 EB93 1F                RAR               restore msb
415 EB94 8F                ADC   A           A=2*A+CY
416 EB95 C9                RET
417                *
418                ************************************************
419                * NORMALIZE CONTENTS BCDE, CORRECT EXPONENT *
420                ************************************************
421                *
422                * Entry: FPT mantissa in BCDE, exponent in MACC.
423                *
424                * Mantissa is normalized and the exponent is
425                * adjusted correctly.
426                *
427 EB96 CDA0EB    L1E191  CALL  :EBA0       Normalize BCDE
428 EB99 D4B7C1            CNC   :C1B7       Add exponents if BCDE<>0
429 EB9C 3F                CMC
430 EB9D 1F                RAR
431 EB9E B7                ORA   A
432 EB9F C9                RET
433                *
434                *
435                *
```

```
                        ********************************
                        * NORMALIZE CONTENTS B,C,D,E *
                        ********************************
                        *
                        * Shifts contents BCDE left until the msb = 1.
                        *
                        * Exit: A: Minus number of shifts.
                        *       HL restored, S+Z-flag set on result.
                        *       CY=1: BCDE was zero.
                        *
EBA0 E5                 L1E192  PUSH  H
EBA1 2E20                       MVI   L,:20     Max 32 bits to shift
EBA3 78                 L1E193  MOV   A,B       Get 1st byte
EBA4 B7                         ORA   A
EBA5 C2BAEB                     JNZ   :EBBA     If '1'-bits in it

                        * Shift 8 bits at once:

EBA8 41                         MOV   B,C       )
EBA9 4A                         MOV   C,D       ) Shift 1 byte
EBAA 53                         MOV   D,E       )
EBAB 5F                         MOV   E,A       )
EBAC 7D                         MOV   A,L
EBAD D608                       SUI   :08       Count minus 8 bits
EBAF 6F                         MOV   L,A
EBB0 C2A3EB                     JNZ   :EBA3     Continu if not ready
EBB3 E1                         POP   H
EBB4 37                         STC             If 4* 8 bits shifted and no
                                                '1' found: BCDE was 0: CY=1
EBB5 C9                         RET

                        * Shift 1 bit:

EBB6 2D                 L1E194  DCR   L         Update count
EBB7 CD48EB                     CALL  :EB48     Shift BCDE 1 bit left
EBBA F2B6EB             L1E195  JP    :EBB6     Again if msb <> 0

                        * If ready:

EBBD 7D                         MOV   A,L       Get nr of shifts left
EBBE D620                       SUI   :20       Calc neg nr of shifts done
EBC0 B7                         ORA   A
EBC1 E1                         POP   H
EBC2 C9                         RET
                        *
                        *********
                        * ROUND *
                        *********
                        *
                        * Rounds up a FPT mantissa in BCD(E). Result in
                        * MACC, exponent also in E.
                        *
                        * Entry: FPT mantissa in BCDE.
                        * Exit:  CY=1: overflow.
                        *        All registers corrupted.
                        *
EBC3 7B                 L1E196  MOV   A,E       Get lobyte mantissa
EBC4 B7                         ORA   A
EBC5 FCD1EB                     CM    :EBD1     Round up BCD if (E)
                                                >= #80
EBC8 D8                         RC              Abort if overflow
EBC9 21D500                     LXI   H,:00D5   Addr MACC
```

```
498 EBCC 5E                 MOV   E,M       Get exp in E
499 EBCD 7E                 MOV   A,M       and in A
500 EBCE C3DBE9             JMP   :E9DB     Copy ABCD into MACC
501                 *
502                 * ROUND UP CONTENTS B,C,D:
503                 *
504                 * Increments a FPT mantissa in BCD with 1 in
505                 * the lsb. If required, the normalized exponent
506                 * is adjusted.
507                 *
508                 * Exit: CY=1: Overflow.
509                 *       AEHL preserved.
510                 *
511 EBD1 14         L1E197 INR   D          Add 1 to lobyte
512 EBD2 C0                RNZ
513 EBD3 0C                INR   C          ) Add 1 to other bits to
514 EBD4 C0                RNZ              ) if overflow
515 EBD5 04                INR   B          )
516 EBD6 C0                RNZ
517 EBD7 0680              MVI   B,:80      If overflow from B: Set B
518                                          for smallest mantissa and
519                                          increment exponent
520                 *
521                 ************************************
522                 * INCREMENT A FPT EXPONENT OF MACC *
523                 ************************************
524                 *
525                 * Exit: ABCDEHL preserved. CY=1: overflow.
526                 *
527 EBD9 C5         L1E198 PUSH  B
528 EBDA F5                PUSH  PSW
529 EBDB E5                PUSH  H
530 EBDC 3E01              MVI   A,:01      1 to be added to exponent
531 EBDE 21D500    L1E199 LXI   H,:00D5    Addr MACC
532 EBE1 CDBAC1            CALL  :C1BA      Add 1 to exponent
533 EBE4 E1                POP   H
534 EBE5 C1                POP   B
535 EBE6 78                MOV   A,B
536 EBE7 C1                POP   B
537 EBE8 C9                RET
538                 *
539                 ************************************************
540                 * DECREMENT FPT EXPONENT OF MACC - (not used) *
541                 ************************************************
542                 *
543 EBE9 C5         L1E274 PUSH  B
544 EBEA F5                PUSH  PSW
545 EBEB E5                PUSH  H
546 EBEC 3EFF              MVI   A,:FF      -1 to be added to exponent
547 EBEE C3DEEB            JMP   :EBDE      Add it to MACC exp
548                 *
549                 ****************************
550                 * TEST IF OPERAND IS ZERO *
551                 ****************************
552                 *
553                 * TSTZA: Test if contents MACC is zero.
554                 * TSTZ:  Test if operand, pointed at by HL, is
555                 *        zero.
556                 *
557                 * Exit: A: hibyte operand.
558                 *       BCDE preserved.
                            HL points to 1st byte of operand.
```

```
560                     *       Z=1; Operand = 0.
561                     *
562 EBF1 21D500         TSTZA   LXI   H,:00D5   Operand = MACC
563 EBF4 7E             TSTZ    MOV   A,M
564 EBF5 23                     INX   H
565 EBF6 B6                     ORA   M
566 EBF7 23                     INX   H
567 EBF8 B6                     ORA   M
568 EBF9 23                     INX   H
569 EBFA B6                     ORA   M         Flags set on result OR
570                                             on all bytes of operand
571 EBFB 2B                     DCX   H
572 EBFC 2B                     DCX   H
573 EBFD 2B                     DCX   H
574 EBFE 7E                     MOV   A,M       Hibyte operand in A
575 EBFF C9                     RET
576                     *
577                     *
578                     *
579 EC00                        END
```

```
****************************
* S Y M B O L   T A B L E *
****************************
```

AADD	EA72	AD10	EA74	AD10A	EB00	ADD11	EB06
ADIV	EA20	AMUL	EA59	ASUB	EA6D	AZERO	EA16
DIV0	EA54	L1E159	EA0E	L1E169	EAB2	L1E170	EAC6
L1E171	EAEF	L1E174	EB0C	L1E175	EB11	L1E176	EB16
L1E177	EB17	L1E180	EB3B	L1E182	EB46	L1E183	EB48
L1E185	EB57	L1E186	EB64	L1E187	EB67	L1E188	EB70
L1E189	EB7D	L1E190	EB82	L1E191	EB96	L1E192	EBA0
L1E193	EBA3	L1E194	EBB6	L1E195	EBBA	L1E196	EBC3
L1E197	EBD1	L1E198	EBD9	L1E199	EBDE	L1E274	EBE9
LSHN	EB39	MDEX	EB1D	OVERF	EA46	OVUNF	EA4B
RSHN	EB55	TSTZ	EBF4	TSTZA	EBF1	UNDRF	EA4E

```
002                             ORG   :EC00
003                     *
004                     *
005                     *
006                     ************************
007                     * FIXED MULTIPLICATION *
008                     ************************
009                     *
010                     * Multiplies a mantissa in registers C,D,E with
011                     * the mantissa of a number in the MACC. The result
012                     * is in B,C,D,E (binary point left of B).
013                     *
014                     * Used for multiplication of mantissa's in a
015                     * FPT multiplication.
016                     *
017                     * Exit: AFHL corrupted.
018                     *
019 EC00 79             MULX    MOV   A,C       )
020 EC01 32DD00                 STA   :00DD     ) Mantissa from CDE into
021 EC04 62                     MOV   H,D       )   00DD, 00DC, 00DB
022 EC05 6B                     MOV   L,E       )
023 EC06 22DB00                 SHLD  :00DB     )
024 EC09 AF                     XRA   A
025 EC0A 57                     MOV   D,A       )
026 EC0B 4F                     MOV   C,A       ) Clear ABCD
027 EC0C 47                     MOV   B,A       )
028 EC0D 3AD800                 LDA   :00D8     Get lobyte MACC mantissa
029 EC10 CD1CEC                 CALL  :EC1C     Multiply
030 EC13 3AD700                 LDA   :00D7     Get next byte MACC mantissa
031 EC16 CD1CEC                 CALL  :EC1C     Multiply
032 EC19 3AD600                 LDA   :00D6     Get hibyte MACC mantissa
033
034                     * Prepare multiplication:
035
036 EC1C 6A             L1E203  MOV   L,D
037 EC1D 59                     MOV   E,C
038 EC1E 50                     MOV   D,B
039 EC1F 47                     MOV   B,A       Byte from MACC in B
040 EC20 AF                     XRA   A
041 EC21 4F                     MOV   C,A
042 EC22 90                     SUB   B
043 EC23 DA29EC                 JC    :EC29     Then multiply
044 EC26 4A                     MOV   C,D
045 EC27 53                     MOV   D,E
046 EC28 C9                     RET
047
048                     * Multiply (product in BDCE):
049
050 EC29 7D             L1E204  MOV   A,L
051 EC2A 8F                     ADC   A
052 EC2B C8                     RZ              Abort if 2*L+CY=0
053 EC2C 6F                     MOV   L,A       Else update L
054 EC2D CD48EB                 CALL  :EB48     Shift BCDE 1 bit left
055 EC30 D229EC                 JNC   :EC29     Again if no overflow
056 EC33 3ADB00                 LDA   :00DB
057 EC36 83                     ADD   E
058 EC37 5F                     MOV   E,A       E=E+(00DB)
059 EC38 3ADC00                 LDA   :00DC
060 EC3B 8A                     ADC   D
061 EC3C 57                     MOV   D,A       D=D+(00DC)+CY
062 EC3D 3ADD00                 LDA   :00DD
063 EC40 89                     ADC   C
```

```
064 EC41 4F                MOV   C,A       C=C+(00DD)+CY
065 EC42 D229EC            JNC   :EC29     Again if no overflow
066 EC45 04                INR   B         If overflow: B=B+1
067 EC46 B7                ORA   A         Clear CY
068 EC47 C329EC            JMP   :EC29     Again
069                *
070                ******************
071                * FIXED DIVISION *
072                ******************
073                *
074                * Divides a mantissa in registers C,D,E by the
075                * mantissa of the number in the MACC. The result
076                * is in B,C,D and the msb of E. The remainder is
077                * in the rest of E and in HL.
078                *
079                * Used to divide mantissa's in a FPT division.
080                *
081                * Exit: AF corrupted.
082                *       CY=1: Overflow in adjusting exponents.
083                *       CY=0: O.K.
084                *
085 EC4A 21D800     DIVX  LXI   H,:00D8   Addr lobyte MACC
086 EC4D 7E                MOV   A,M       )
087 EC4E 93                SUB   E         )
088 EC4F 77                MOV   M,A       )
089 EC50 2B                DCX   H         ) Mantissa MACC =
090 EC51 7E                MOV   A,M       ) CDE - mantissa MACC
091 EC52 9A                SBB   D         )
092 EC53 77                MOV   M,A       )
093 EC54 2B                DCX   H         )
094 EC55 7E                MOV   A,M       )
095 EC56 99                SBB   C         )
096 EC57 77                MOV   M,A       )
097 EC58 21DD00            LXI   H,:00DD   Addr save area
098 EC5B 37                STC
099 EC5C 79                MOV   A,C       )
100 EC5D 1F                RAR             )
101 EC5E 77                MOV   M,A       )
102 EC5F 2B                DCX   H         )
103 EC60 7A                MOV   A,D       ) 00DD,00DC,00DB =
104 EC61 1F                RAR             ) CDE SHR 1 with msb C=1
105 EC62 77                MOV   M,A       )
106 EC63 2B                DCX   H         )
107 EC64 7B                MOV   A,E       )
108 EC65 1F                RAR             )
109 EC66 77                MOV   M,A       )
110 EC67 2B                DCX   H
111 EC68 0600              MVI   B,:00
112 EC6A 78                MOV   A,B       )
113 EC6B 1F                RAR             )
114 EC6C 77                MOV   M,A       ) 00DA =00 or 80, depending
115                                        ) on result RAR
116 EC6D 21D600            LXI   H,:00D6
117 EC70 7E                MOV   A,M       )
118 EC71 23                INX   H         ) Get mantissa MACC in ADE
119 EC72 56                MOV   D,M       )
120 EC73 23                INX   H         )
121 EC74 5E                MOV   E,M       )
122 EC75 B7                ORA   A
123 EC76 FAC4EC            JM    :ECC4     Jump if normalised
124 EC79 CDD9EB            CALL  :EBD9     Incr FPT exponent
125 EC7C D8                RC              Abort if overflow
```

```
126 EC7D 6B                 MOV   L,E       )
127 EC7E 62                 MOV   H,D       ) Remainder in EHL
128 EC7F 5F                 MOV   E,A       )
129 EC80 1601               MVI   D,:01
130 EC82 48                 MOV   C,B
131 EC83 C5         L1E206  PUSH  B
132 EC84 44                 MOV   B,H
133 EC85 4D                 MOV   C,L
134 EC86 21DA00             LXI   H,:00DA
135 EC89 AF                 XRA   A
136 EC8A 96                 SUB   M
137 EC8B 23                 INX   H
138 EC8C 79                 MOV   A,C
139 EC8D 9E                 SBB   M
140 EC8E 4F                 MOV   C,A
141 EC8F 23                 INX   H
142 EC90 78                 MOV   A,B
143 EC91 9E                 SBB   M
144 EC92 47                 MOV   B,A
145 EC93 23                 INX   H
146 EC94 7B                 MOV   A,E
147 EC95 9E                 SBB   M
148 EC96 5F                 MOV   E,A
149 EC97 69                 MOV   L,C
150 EC98 60                 MOV   H,B
151 EC99 C1                 POP   B
152 EC9A 3ADA00     L1E207  LDA   :00DA
153 EC9D 07                 RLC
154 EC9E 78                 MOV   A,B
155 EC9F 17                 RAL
156 ECA0 3F                 CMC
157 ECA1 D0                 RNC
158 ECA2 1F                 RAR
159 ECA3 7D                 MOV   A,L
160 ECA4 17                 RAL
161 ECA5 6F                 MOV   L,A
162 ECA6 7C                 MOV   A,H
163 ECA7 17                 RAL
164 ECA8 67                 MOV   H,A
165 ECA9 CD48EB             CALL  :EB48     Shift BCDE left 1 bit
166 ECAC 7A                 MOV   A,D
167 ECAD 0F                 RRC
168 ECAE DA83EC             JC    :EC83
169 ECB1 C5         L1E208  PUSH  B
170 ECB2 44                 MOV   B,H
171 ECB3 4D                 MOV   C,L
172 ECB4 2ADB00             LHLD  :00DB
173 ECB7 09                 DAD   B
174 ECB8 3ADD00             LDA   :00DD
175 ECBB 8B                 ADC   E
176 ECBC 5F                 MOV   E,A
177 ECBD C1                 POP   B
178 ECBE 3ADA00             LDA   :00DA
179 ECC1 C39DEC             JMP   :EC9D
180 ECC4 6B         L1E209  MOV   L,E
181 ECC5 62                 MOV   H,D
182 ECC6 5F                 MOV   E,A
183 ECC7 50                 MOV   D,B
184 ECC8 48                 MOV   C,B
185 ECC9 C3B1EC             JMP   :ECB1
186                 *
```

```
                        ************************************
                        * AMD: ISSUE COMMAND TO MATH.CHIP *
                        ************************************
                        *
                        * Entry: HL points to command.
                        * Exit:  HL updated, A corrupted, BCDEF preserved.
                        *
ECCC 7E         MPT15   MOV   A,M       Get command
ECCD 23                 INX   H
ECCE 3202FB             STA   :FB02     Issue cmd to math.chip
ECD1 C9                 RET
                        *
                        ******************************
                        * AMD: TURN OFF ERROR STATUS *
                        ******************************
                        *
                        * Exit: All registers preserved.
                        *
ECD2 F5         MPT16   PUSH  PSW
ECD3 AF                 XRA   A
ECD4 3202FB             STA   :FB02     Cmd math.chip = 0
ECD7 F1                 POP   PSW
ECD8 C9                 RET
                        *
                        ******************************************
                        * AMD: LOAD 16-BIT DATA INTO MATH.CHIP *
                        ******************************************
                        *
                        * Entry: 1st byte in A, 2nd on stack.
                        *
ECD9 3200FB     MPT17   STA   :FB00     Load 1st byte in math.chip
ECDC F1                 POP   PSW
ECDD 3200FB             STA   :FB00     Load data in math.chip
ECE0 C9                 RET
                        *
                        ***************************
                        * part of 'IAND' (1E32C) *
                        ***************************
                        *
ECE1 CD8CE3     L1E213  CALL  :E38C     Copy MACC into EBCA
ECE4 CD35E3             CALL  :E335     Run IAND
ECE7 C385E3             JMP   :E385     Copy ABCD into MACC
                        *
                        ***************************
                        * part of 'IOR' (1E345) *
                        ***************************
                        *
ECEA CD8CE3     L1E214  CALL  :E38C     Copy MACC into EBCA
ECED CD4CE3             CALL  :E34C     Run IOR
ECF0 C385E3             JMP   :E385     Copy ABCD into MACC
                        *
                        ***************************
                        * part of 'IXOR' (1E35C) *
                        ***************************
                        *
ECF3 CD8CE3     L1E215  CALL  :E38C     Copy MACC into EBCA
ECF6 CD63E3             CALL  :E363     Run IXOR
ECF9 C385E3             JMP   :E385     Copy ABCD into MACC
                        *
                        ******************************************
                        * COPY MACC INTO REGISTERS A,B,C,D; CMA *
                        ******************************************
```

```
                        *
                        * Part of 1E373.
                        *
ECFC CD33E1     L1E216  CALL  :E133     Copy MACC into ABCD
ECFF 2F                 CMA             Compl exponent byte
ED00 C9                 RET
                        *
                        ************************************
                        * COPY MACC INTO REGISTERS E,B,C,A *
                        ************************************
                        *
                        * Part of 1E38C.
                        *
ED01 CD33E1     L1E217  CALL  :E133     Copy MACC into ABCD
ED04 5F         IGP10   MOV   E,A       Exp in E
ED05 C38FE3             JMP   :E38F     Copy BCDE into EBCA
                        *
                        ************************************
                        * COPY MACC INTO REGISTERS B,C,D,E *
                        ************************************
                        *
                        * Part of SHR (1E398) and SHL (1E3A5).
                        * Tests if the value of a INT operand in memory is
                        * bigger than 32 (nr of bits for a mantissa).
                        * If not, the contents of the MACC is copied
                        * into the registers BCDE. If the number is too
                        * big, the registers BCDE are cleared.
                        *
                        * Entry: HL points to INT operand in memory.
                        *
ED08 CDB2E3     L1E219  CALL  :E3B2     Test if operand > 31; if
                                        true: clear ABCDE
ED0B CCD6E3             CZ    :E3D6     If OK: nr in A, contents
                                        MACC into BCDE
ED0E C9                 RET
                        *
                        ************************************************
                        * COPY CONTENTS MACC INTO REGISTERS B,C,D,E *
                        ************************************************
                        *
                        * Entry L1E220: Not used.
                        * Entry GBC10 : Copy ABCD into BCDE.
                        *
ED0F F5         L1E220  PUSH  PSW
ED10 CD6FE5             CALL  :E56F     Copy MACC into ABCD
                        *
ED13 5A         GBC10   MOV   E,D       )
ED14 51                 MOV   D,C       ) Copy ABCD into BCDE
ED15 48                 MOV   C,B       )
ED16 47                 MOV   B,A       )
ED17 F1                 POP   PSW
ED18 C9                 RET
                        *
                        *************
                        * AMD: IAND *
                        *************
                        *
                        * MTOS = MTOS IAND MEM.
                        *
                        * Entry: HL points to operand in memory.
                        * Exit:  All registers preserved.
                        *
```

```
312 ED19 F5          ZIAND   PUSH  PSW
313 ED1A C5                  PUSH  B
314 ED1B D5                  PUSH  D
315 ED1C E5                  PUSH  H
316 ED1D CD4FED              CALL  :ED4F     Copy MTOS into EBCA
317 ED20 CD35E3              CALL  :E335     Run IAND
318 ED23 C33DED              JMP   :ED3D     Result into MTOS
319                  *
320                  ************
321                  * AMD: IOR *
322                  ************
323                  *
324                  * MTOS = MTOS IOR MEM.
325                  *
326                  * Entry: HL points to operand in memory.
327                  * Exit:  All registers preserved.
328                  *
329 ED26 F5          ZIOR    PUSH  PSW
330 ED27 C5                  PUSH  B
331 ED28 D5                  PUSH  D
332 ED29 E5                  PUSH  H
333 ED2A CD4FED              CALL  :ED4F     Copy MTOS into EBCA
334 ED2D CD4CE3              CALL  :E34C     Run IOR
335 ED30 C33DED              JMP   :ED3D     Result into MTOS
336                  *
337                  *************
338                  * AMD: IXOR *
339                  *************
340                  *
341                  * MTOS = MTOS IXOR MEM.
342                  *
343                  * Entry: HL points to operand in memory.
344                  * Exit:  All registers preserved.
345                  *
346 ED33 F5          ZIXOR   PUSH  PSW
347 ED34 C5                  PUSH  B
348 ED35 D5                  PUSH  D
349 ED36 E5                  PUSH  H
350 ED37 CD4FED              CALL  :ED4F     Copy MTOS into EBCA
351 ED3A CD63E3              CALL  :E363     Run IXOR; result in ABCD
352 ED3D CD5FE5      ZORT1   CALL  :E55F     Copy ABCD into MTOS
353 ED40 C34DC1              JMP   :C14D     Popall, ret
354                  *
355                  *************
356                  * AMD: INOT *
357                  *************
358                  *
359                  * MTOS = INOT (MTOS).
360                  *
361                  * Exit: All registers preserved.
362                  *
363                  * REMARK: Wrong routine: MTOS is made -MTOS,
364                  *         and then 1 is added. So result is
365                  *         INOT (MTOS)+2.
366                  *         Correct would be: Add -1.
367                  *
368 ED43 CD22E5      ZINOT   CALL  :E522     Change sign MTOS (INT)
369 ED46 E5                  PUSH  H
370 ED47 2120C4              LXI   H,:C420   Addr INT (1)
371 ED4A CDF4E4              CALL  :E4F4     MTOS = - MTOS + 1
372 ED4D E1                  POP   H
373 ED4E C9                  RET
```

```
374                     *
375                     ****************************************
376                     * AMD: COPY MTOS INTO REGISTERS E,B,C,A *
377                     ****************************************
378                     *
379 ED4F CD6FE5      ZIGTP   CALL  :E56F     Copy MTOS into ABCD
380 ED52 C304ED             JMP   :ED04     Copy ABCD into EBCA
381                     *
382                     ************
383                     * AMD: SHR *
384                     ************
385                     *
386                     * Shifts MTOS right MEM positions.
387                     *
388                     * Entry: HL points to INT number in memory.
389                     * Exit:  All registers preserved.
390                     *
391 ED55 F5          ZSHR    PUSH  PSW
392 ED56 C5                  PUSH  B
393 ED57 D5                  PUSH  D
394 ED58 E5                  PUSH  H
395 ED59 CDB2E3              CALL  :E3B2     Check value of MEM. Value in
396                                          A. Clear ABCDE if too big
397 ED5C CC7CED              CZ    :ED7C     Else: Copy MTOS in BCDE
398 ED5F CD55EB              CALL  :EB55     Shift BCDE right A positions
399 ED62 78          ZRREG   MOV   A,B       )
400 ED63 41                  MOV   B,C       )
401 ED64 4A                  MOV   C,D       ) Copy BCDE into ABCD
402 ED65 53                  MOV   D,E       )
403 ED66 CD5FE5              CALL  :E55F     Copy ABCD into MTOS
404 ED69 C34DC1              JMP   :C14D     Popall, ret
405                     *
406                     ************
407                     * AMD: SHL *
408                     ************
409                     *
410                     * Shifts MTOS left MEM positions.
411                     *
412                     * Entry: HL points to INT number in memory.
413                     * Exit:  All registers preserved.
414                     *
415 ED6C F5          ZSHL    PUSH  PSW
416 ED6D C5                  PUSH  B
417 ED6E D5                  PUSH  D
418 ED6F E5                  PUSH  H
419 ED70 CDB2E3              CALL  :E3B2     Test value of MEM. Value in
420                                          A. Clear ABCDE if too big
421 ED73 CC7CED              CZ    :ED7C     If OK: Copy MTOS into BCDE
422 ED76 CD39EB              CALL  :EB39     Shift BCDE left A positions
423 ED79 C362ED              JMP   :ED62     Copy BCDE into MTOS
424                     *
425                     ****************************************
426                     * AMD: COPY MTOS INTO REGISTERS B,C,D,E *
427                     ****************************************
428                     *
429                     * Exit: AHL preserved.
430                     *
431 ED7C F5          ZGBCDE  PUSH  PSW
432 ED7D CD6FE5              CALL  :E56F     Copy MTOS into ABCD
433 ED80 C313ED              JMP   :ED13     Copy ABCD into BCDE
434                     *
435                     *
```

```
                        ************************************************
                        * CHECK IF CONTENTS REGISTERS B,C,D,E IS ZERO *
                        ************************************************
                        *
                        * Exit: Z=1: Contents BCDE is zero.
                        *       BCDEHL preserved.
                        *
ED83 78         L1E232  MOV   A,B
ED84 B1                 ORA   C
ED85 B2                 ORA   D
ED86 B3                 ORA   E
ED87 C9                 RET
                        *
                        ******************************************
                        * EVT. NEGATE CONTENTS REGISTERS B,C,D,E *
                        ******************************************
                        *
                        * If S=1: Contents BCDE is negated. Overflow
                        *         exit if negation not possible.
                        * On exit, flags are set on contents entry A.
                        *
ED88 F5         L1E233  PUSH  PSW
ED89 FCC9E3             CM    :E3C9     Evt negate BCDE
ED8C F1                 POP   PSW
ED8D B7                 ORA   A
ED8E C9                 RET
                        *
                        ****************
                        * SIGN COMPARE *
                        ****************
                        *
                        * Entry: HL: Points to divisor.
                        * Exit:  B:  Exponent MACC.
                        *        F:  Set on XOR of exp bytes MACC and MEM.
                        *            S=1 if difference in sign.
                        *
ED8F 3AD500     L1E234  LDA   :00D5     Get exp byte MACC
ED92 47                 MOV   B,A       in B
ED93 AE                 XRA   M         XOR with exp byte MEM
ED94 C9                 RET
                        *
                        ************************************
                        * AMD: GET STATUS BITS MATH.CHIP *
                        ************************************
                        *
                        * Exit: A: Status.
                        *       FBCDEHL preserved.
                        *
ED95 CD2DE5     M4STAT  CALL  :E52D     Operate immediate
ED98 37                 DATA  :37       Push MTOS
ED99 CD2DE5             CALL  :E52D     Operate immediate
ED9C 38                 DATA  :38       Pop MTOS
ED9D 3A02FB             LDA   :FB02     Get status math.chip
EDA0 C9                 RET
                        *
                        **************
                        * AMD: POWER *
                        **************
                        *
                        * MTOS = MTOS ^ MEM.
                        *
                        * Entry: HL points to power in memory.
```

```
                        * Exit:  AF corrupted, BCDEHL preserved.
                        *
EDA1 CD95ED             ZPWR    CALL  :ED95     Get status math.chip
EDA4 E620               ANI   :20       MTOS empty ?
EDA6 C0                 RNZ             Abort if not
EDA7 C3ACE4             JMP   :E4AC     Run PWR routine
                        *
                        ****************
                        * FPT ADDITION *
                        ****************
                        *
                        * For FPT values: MACC = MACC + MEM.
                        *
                        * Entry: HL points to FPT number in memory.
                        * Exit:  All registers preserved.
                        *
EDAA F5                 XFADD   PUSH  PSW
EDAB C5                 PUSH  B
EDAC D5                 PUSH  D
EDAD E5                 PUSH  H
EDAE CD72EA             CALL  :EA72     MACC = MACC + MEM
EDB1 C34DC1             JMP   :C14D     Popall, ret
                        *
                        ********************
                        * FPT SUBTRACTION *
                        ********************
                        *
                        * For FPT values: MACC = MACC - MEM.
                        *
                        * Entry: HL points to FPT number in memory.
                        * Exit:  All registers preserved.
                        *
EDB4 F5                 XFSUB   PUSH  PSW
EDB5 C5                 PUSH  B
EDB6 D5                 PUSH  D
EDB7 E5                 PUSH  H
EDB8 CD6DEA             CALL  :EA6D     MACC = MACC - MEM
EDBB C34DC1             JMP   :C14D     Popall, ret
                        *
EDBE FF                 DATA  :FF
EDBF FF                 DATA  :FF
                        *
                        ****************************************
                        * SAVEA: PREPARE SAVING STRING ARRAYS *
                        ****************************************
                        *
                        * Reserves space in free RAM for a string, composed
                        * from all string elements of a string array.
                        * The array elements are moved into this area.
                        * If not sufficient free RAM available, 'OUT OF
                        * MEMORY' error occurs.
                        *
                        * Entry: DE: Length array to be saved.
                        *        HL: Pointer to array.
                        * Exit:  DE: Length of block in free RAM.
                        *        HL: Startaddress block in free RAM.
                        *        AF corrupted, BC preserved.
                        *
EDC0 C5                 MSA     PUSH  B
EDC1 42                 MOV   B,D       ) Length array in BC
EDC2 4B                 MOV   C,E       )
EDC3 EB                 XCHG            Varptr in DE
```

```
560 EDC4 2AA302             LHLD  :02A3     Get startaddr free RAM spac
561 EDC7 E5                 PUSH  H         and save it on stack
562 EDC8 EB                 XCHG            and in DE; HL is array pntr
563 EDC9 78                 MOV   A,B       )
564 EDCA CDFDED             CALL  :EDFD     ) Save length array in 1st
565 EDCD 79                 MOV   A,C       ) location free RAM
566 EDCE CDFDED             CALL  :EDFD     )
567 EDD1 78         L1E240  MOV   A,B
568 EDD2 B1                 ORA   C
569 EDD3 CAE5ED             JZ    :EDE5     Abort if ready
570 EDD6 7E                 MOV   A,M
571 EDD7 23                 INX   H
572 EDD8 E5                 PUSH  H         Addr array pntr on stack
573 EDD9 66                 MOV   H,M       ) Addr string element in HL
574 EDDA 6F                 MOV   L,A       )
575 EDDB CDEDED             CALL  :EDED     Store element in free RAM
576 EDDE E1                 POP   H         Get array pntr back
577 EDDF 23                 INX   H         Pnts to next element
578 EDE0 0B                 DCX   B         ) Decr length still to be
579 EDE1 0B                 DCX   B         ) done
580 EDE2 C3D1ED             JMP   :EDD1     Continu
581
582                 * If ready:
583
584 EDE5 E1        L1E241  POP   H         Get startaddr new string
585 EDE6 EB                 XCHG            in DE; HL is end used area
586 EDE7 CD1ADE             CALL  :DE1A     Calc length of string
587 EDEA EB                 XCHG            in DE
588 EDEB C1                 POP   B
589 EDEC C9                 RET
590                 *
591                 * COPY STRING ELEMENT INTO FREE RAM:
592                 *
593 EDED C5        L1E242  PUSH  B
594 EDEE 46                 MOV   B,M       Get length of string in B
595 EDEF 7E        L1E243  MOV   A,M       Byte in A
596 EDF0 23                 INX   H         Pnts to next byte
597 EDF1 CDFDED             CALL  :EDFD     Copy byte into free RAM
598 EDF4 78                 MOV   A,B       )
599 EDF5 D601               SUI   :01       ) Update length
600 EDF7 47                 MOV   B,A       )
601 EDF8 D2EFED             JNC   :EDEF     Next byte if not ready
602 EDFB C1                 POP   B
603 EDFC C9                 RET
604                 *
605                 * STORE STRINGDATA IN FREE RAM SPACE:
606                 *
607                 * Moves 1 byte of a string array element into the
608                 * free RAM space and checks for 'OUT OF MEMORY'.
609                 *
610                 * Entry: DE: Points to 1st free address in RAM.
611                 *         A:  Byte to be moved.
612                 * Exit:  DE updated, BCHL preserved.
613                 *
614 EDFD 12        L1E244  STAX  D         Store byte in free RAM
615 EDFE 13                 INX   D         Update RAM pntr
616 EDFF E5                 PUSH  H
617 EE00 2AA502             LHLD  :02A5     Get addr bottom screen RAM
618 EE03 CD14DE             CALL  :DE14     End free RAM reached ?
619 EE06 DA10DA             JC    :DA10     Then run error 'OUT OF
620                                          MEMORY'
621 EE09 E1                 POP   H
```

```
622 EE0A C9                 RET
623                 *
624                 *
625                 *
626 EE0B                    END
```

```
*****************************
* S Y M B O L   T A B L E *
*****************************
```

DIVX	EC4A	GBC10	ED13	IGP10	ED04	L1E203	EC1C
L1E204	EC29	L1E206	EC83	L1E207	EC9A	L1E208	ECB1
L1E209	ECC4	L1E213	ECE1	L1E214	ECEA	L1E215	ECF3
L1E216	ECFC	L1E217	ED01	L1E219	ED08	L1E220	ED0F
L1E232	ED83	L1E233	ED88	L1E234	ED8F	L1E240	EDD1
L1E241	EDE5	L1E242	EDED	L1E243	EDEF	L1E244	EDFD
M4STAT	ED95	MPT15	ECCC	MPT16	ECD2	MPT17	ECD9
MSA	EDC0	MULX	EC00	XFADD	EDAA	XFSUB	EDB4
ZGBCDE	ED7C	ZIAND	ED19	ZIGTP	ED4F	ZINOT	ED43
ZIOR	ED26	ZIXOR	ED33	ZORT1	ED3D	ZPWR	EDA1
ZRREG	ED62	ZSHL	ED6C	ZSHR	ED55		

```
002                         ORG   :EE0B
003                 *
004                 *
005                 *
006                 **************
007                 * (not used) *
008                 **************
009                 *
010 EE0B E5         L1E245  PUSH  H
011 EE0C CD91D8             CALL  :D891
012                 *
013                 ************************************************
014                 * LOADA: READ ARRAY DATA FROM TAPE INTO ARRAY *
015                 ************************************************
016                 *
017                 * Reads block(s) from tape into the free RAM space
018                 * and move it afterwards into the array.
019                 *
020                 * Entry: HL: Length of name requested.
021                 *         A:  Variable type.
022                 *         On stack: Address where to dump data.
023                 *
024 EE0F C5         L1E273  PUSH  B
025 EE10 F5                 PUSH  PSW       Save var.type
026 EE11 010032             LXI   B,:3200   File type in B, C for load
027                                         during program
028 EE14 CD81D6             CALL  :D681     Open read file
029 EE17 E1                 POP   H         Get var.type in H
030 EE18 CDB5EF             CALL  :EFB5     Read var.type from tape
031 EE1B BC                 CMP   H         Correct type ?
032 EE1C C21ADA             JNZ   :DA1A     Run error 'TYPE MISMATCH'
033                                         if not
034 EE1F FE20               CPI   :20
035 EE21 CA38EE             JZ    :EE3B     Jump if string arrays
036
037                 * If INT/FPT arrays:
038
039 EE24 E1                 POP   H
040 EE25 E3                 XTHL            Get dumpaddr in HL
041 EE26 EB                 XCHG            Get length in HL
042 EE27 19                 DAD   D         HL is end dump area
043 EE28 EB                 XCHG            in DE; HL is start dump
044 EE29 CDD102             CALL  :02D1     Read block from tape
045 EE2C D2B3D2             JNC   :D2B3     Evt run 'LOADING ERROR ..'
046 EE2F CDD402     L1E246  CALL  :02D4     Stop reading
047 EE32 3A4000             LDA   :0040     Get POROM
048 EE35 C3A6EF             JMP   :EFA6     Select ROM bank 0, abort
049
050                 * If string arrays:
051
052 EE38 D5         L1E247  PUSH  D
053 EE39 2AA502             LHLD  :02A5     Get bottom screen RAM
054 EE3C EB                 XCHG            in DE
055 EE3D 2AA302             LHLD  :02A3     Get begin free RAM space
056 EE40 E5                 PUSH  H
057 EE41 CD97D8             CALL  :D897     Read block from tape into
058                                         free RAM area; evt.run error
059 EE44 E1                 POP   H         Get begin free RAM
060 EE45 56                 MOV   D,M       )
061 EE46 23                 INX   H         ) Length of array in DE
062 EE47 5E                 MOV   E,M       )
063 EE48 23                 INX   H
```

```
064 EE49 C3E5D6             JMP   :D6E5     Via D6E5 to 1EE4C
065                 *
066 EE4C C5         L1E248  PUSH  B         Save length array
067 EE4D D5                 PUSH  D
068 EE4E CDA8CB             CALL  :CBA8     Erase stringreference
069                                         in heap and symtab
070 EE51 2B                 DCX   H
071 EE52 2B                 DCX   H
072 EE53 D1                 POP   D
073 EE54 E5                 PUSH  H
074 EE55 1A                 LDAX  D
075 EE56 CD8BD1             CALL  :D18B     Get place in heap for string
076 EE59 E5                 PUSH  H
077 EE5A CD72D1             CALL  :D172     Transfer string into heap
078 EE5D C1                 POP   B
079 EE5E E1                 POP   H         )
080 EE5F 71                 MOV   M,C       ) Length into heap at
081 EE60 23                 INX   H         ) begin of string
082 EE61 70                 MOV   M,B       )
083 EE62 23                 INX   H
084 EE63 C1                 POP   B         Get length array
085 EE64 0B                 DCX   B         ) Update it
086 EE65 0B                 DCX   B         )
087 EE66 78                 MOV   A,B
088 EE67 B1                 ORA   C
089 EE68 C24CEE             JNZ   :EE4C     Next string if not ready
090 EE6B C32FEE             JMP   :EE2F     Stop reading, select ROM
091                                         bank 0; abort
092                 *
093                 *
094                 *   ============
095                 *** SOUND MODULE ***
096                 *   ============
097                 *
098                 *
099 EE6E 0E00       TEMPO   MVI   C,:00     Count SCB
100 EE70 21C201             LXI   H,:01C2   Addr sound control block 0
101 EE73 E5         L1E250  PUSH  H         Preserve addr SCB
102 EE74 7E                 MOV   A,M       Get value of volume counter
103 EE75 FEFE               CPI   :FE
104 EE77 CA7EEE             JZ    :EE7E     If FE: No increment (sound
105                                         forever)
106 EE7A D29DEF             JNC   :EF9D     If FF: Goto next block
107                                         (sound off)
108 EE7D 34                 INR   M         ) Incr duration count volume
109 EE7E 3C         L1E251  INR   A         )
110 EE7F E5                 PUSH  H         Preserve addr SCB
111 EE80 47                 MOV   B,A       Save incr duration count
112 EE81 23                 INX   H
113 EE82 5E                 MOV   E,M       ) Get pntr envelope count
114 EE83 23                 INX   H         ) in DE
115 EE84 56                 MOV   D,M       )
116 EE85 1A                 LDAX  D         Get envelope duration count
117 EE86 B8                 CMP   B         Comp with volume count
118 EE87 D2B8EE             JNC   :EEB8     Jump if env. not counted out
119
120                 * Envelope counted out:
121
122 EE8A EB                 XCHG
123 EE8B E3                 XTHL            Addr env.duration on stack;
124                                         addr SCB in HL
125 EE8C 3600               MVI   M,:00     Present duration count is 0
```

```
064 EE49 C3E5D6             JMP   :D6E5     Via D6E5 to 1EE4C
065                 *
066 EE4C C5         L1E248  PUSH  B         Save length array
067 EE4D D5                 PUSH  D
068 EE4E CDA8CB             CALL  :CBA8     Erase stringreference
069                                         in heap and symtab
070 EE51 2B                 DCX   H
071 EE52 2B                 DCX   H
072 EE53 D1                 POP   D
073 EE54 E5                 PUSH  H
074 EE55 1A                 LDAX  D
075 EE56 CD8BD1             CALL  :D18B     Get place in heap for string
076 EE59 E5                 PUSH  H
077 EE5A CD72D1             CALL  :D172     Transfer string into heap
078 EE5D C1                 POP   B
079 EE5E E1                 POP   H         )
080 EE5F 71                 MOV   M,C       ) Length into heap at
081 EE60 23                 INX   H         ) begin of string
082 EE61 70                 MOV   M,B       )
083 EE62 23                 INX   H
084 EE63 C1                 POP   B         Get length array
085 EE64 0B                 DCX   B         ) Update it
086 EE65 0B                 DCX   B         )
087 EE66 78                 MOV   A,B
088 EE67 B1                 ORA   C
089 EE68 C24CEE             JNZ   :EE4C     Next string if not ready
090 EE6B C32FEE             JMP   :EE2F     Stop reading, select ROM
091                                         bank 0; abort
092                 *
093                 *
094                 *   ============
095                 *** SOUND MODULE ***
096                 *   ============
097                 *
098                 *
099 EE6E 0E00       TEMPO   MVI   C,:00     Count SCB
100 EE70 21C201             LXI   H,:01C2   Addr sound control block 0
101 EE73 E5         L1E250  PUSH  H         Preserve addr SCB
102 EE74 7E                 MOV   A,M       Get value of volume counter
103 EE75 FEFE               CPI   :FE
104 EE77 CA7EEE             JZ    :EE7E     If FE: No increment (sound
105                                         forever)
106 EE7A D29DEF             JNC   :EF9D     If FF: Goto next block
107                                         (sound off)
108 EE7D 34                 INR   M         ) Incr duration count volume
109 EE7E 3C         L1E251  INR   A         )
110 EE7F E5                 PUSH  H         Preserve addr SCB
111 EE80 47                 MOV   B,A       Save incr duration count
112 EE81 23                 INX   H
113 EE82 5E                 MOV   E,M       ) Get pntr envelope count
114 EE83 23                 INX   H         ) in DE
115 EE84 56                 MOV   D,M       )
116 EE85 1A                 LDAX  D         Get envelope duration count
117 EE86 B8                 CMP   B         Comp with volume count
118 EE87 D2B8EE             JNC   :EEB8     Jump if env. not counted out
119
120                 * Envelope counted out:
121
122 EE8A EB                 XCHG
123 EE8B E3                 XTHL            Addr env.duration on stack;
124                                         addr SCB in HL
125 EE8C 3600               MVI   M,:00     Present duration count is 0
```

```
126 EE8E E1                 POP   H         Get pntr to env.table
127 EE8F 23                 INX   H         +1
128 EE90 7E                 MOV   A,M       Get next env. duration
129 EE91 B7                 ORA   A         Is it FF ?
130 EE92 FC93EF             CM    :EF93     Then restart envelope
131 EE95 47                 MOV   B,A       Env duration in B
132 EE96 23                 INX   H         Pnts to next pos env table
133 EE97 7E                 MOV   A,M       Value in A
134 EE98 EB                 XCHG
135 EE99 72                 MOV   M,D       ) Set env pointer in SCB
136 EE9A 2B                 DCX   H         ) to new time field
137 EE9B 73                 MOV   M,E       )
138 EE9C 23                 INX   H
139 EE9D 23                 INX   H
140 EE9E 23                 INX   H
141 EE9F 23                 INX   H         HL pnts to vol.multiplier
142 EEA0 E5                 PUSH  H
143 EEA1 6E                 MOV   L,M       Sound volume *8 in L
144 EEA2 2600               MVI   H,:00
145 EEA4 29                 DAD   H         *16 in HL
146 EEA5 EB                 XCHG            Multiplier in DE
147 EEA6 218000             LXI   H,:0080   Init.value
148 EEA9 05         L1E252  DCR   B         Decr. envelope duration
149 EEAA FAB1EE             JM    :EEB1     Jump if ready
150 EEAD 19                 DAD   D         Add multiplier
151 EEAE C3A9EE             JMP   :EEA9     Again
152 EEB1 44         L1E253  MOV   B,H       New eff. volume in B
153 EEB2 E1                 POP   H
154 EEB3 23                 INX   H
155 EEB4 70                 MOV   M,B       Set new basic volume
156 EEB5 C3BDEE             JMP   :EEBD
157
158                 * If envelope not counted out:
159
160 EEB8 D1         L1E254  POP   D
161 EEB9 23                 INX   H
162 EEBA 23                 INX   H
163 EEBB 23                 INX   H
164 EEBC 23                 INX   H
165
166                 * Handle tremolo:
167
168 EEBD 46         L1E255  MOV   B,M       Get basic volume in B
169 EEBE 23                 INX   H
170 EEBF 7E                 MOV   A,M       Get tremolo count
171 EEC0 B7                 ORA   A
172 EEC1 CAEBEE             JZ    :EEEB     Jump if no tremolo adj.
173 EEC4 C601               ADI   :01       )
174 EEC6 CE00               ACI   :00       ) Incr tremolo count
175 EEC8 77                 MOV   M,A       )
176 EEC9 1F                 RAR
177 EECA 1F                 RAR
178 EECB 1F                 RAR
179 EECC D2DCEE             JNC   :EEDC     No adj. if bit 2 of
180                                         <T> is 0
181 EECF 04                 INR   B         )
182 EED0 04                 INR   B         ) Else add 4 units to
183 EED1 04                 INR   B         ) basic volume
184 EED2 04                 INR   B         )
185 EED3 1F                 RAR
186 EED4 00                 NOP
187 EED5 DADCEE             JC    :EEDC     No adjust if bit 2 of
```

```
188                                                  <T> = 0
189 EED8 78                  MOV   A,B
190 EED9 D608                SUI   :08              Else: basic vol. -2 units
191 EEDB 47                  MOV   B,A
192                  *
193 EEDC 00          L1E256  NOP
194 EEDD 78                  MOV   A,B              Get updated basic volume
195 EEDE 0600                MVI   B,:00
196 EEE0 B7                  ORA   A
197 EEE1 FAEBEE              JM    :EEEB            Jump if 81-FF
198 EEE4 060F                MVI   B,:0F
199 EEE6 B8                  CMP   B
200 EEE7 D2EBEE              JNC   :EEEB            Jump if >=0F (max.value)
201 EEEA 47                  MOV   B,A
202                  *
203 EEEB 23          L1E257  INX   H                Pnts to actual volume
204 EEEC 78                  MOV   A,B              Get new basic volume
205 EEED 96                  SUB   M                Minus actual volume
206 EEEE 07                  RLC                    )
207 EEEF 1F                  RAR                    )
208 EEF0 3F                  CMC                    ) New actual volume is
209 EEF1 CE00                ACI   :00              ) old one + 0.5
210 EEF3 07                  RLC                    ) (difference +/- 1)
211 EEF4 1F                  RAR                    )
212 EEF5 1F                  RAR                    )
213 EEF6 86                  ADD   M                )
214 EEF7 77                  MOV   M,A              Store in SCB
215 EEF8 47                  MOV   B,A              and in B
216 EEF9 E5                  PUSH  H
217 EEFA C5                  PUSH  B
218 EEFB 119402              LXI   D,:0294          Addr POROM
219 EEFE 2104FD              LXI   H,:FD04          Addr PORO
220 EF01 79                  MOV   A,C              Get SCB count
221 EF02 0EF0                MVI   C,:F0            Mask for vol. SCB0,SCB2
222 EF04 0F                  RRC
223 EF05 D212EF              JNC   :EF12            Jump if SCB0,SCB2
224 EF08 0E0F                MVI   C,:0F            Mask for vol. SCB1,NCB
225 EF0A F5                  PUSH  PSW
226 EF0B 78                  MOV   A,B              )
227 EF0C 87                  ADD   A                ) Actual volume into
228 EF0D 87                  ADD   A                ) hinibble for SCB1
229 EF0E 87                  ADD   A                ) and NCB
230 EF0F 87                  ADD   A                )
231 EF10 47                  MOV   B,A              )
232 EF11 F1                  POP   PSW
233 EF12 1F          L1E258  RAR
234 EF13 D218EF              JNC   :EF18            Jump if SCB0,SCB1
235 EF16 13                  INX   D                ) POROM+1,PORO+1
236 EF17 23                  INX   H                ) for SCB2,NCB
237 EF18 1A          L1E259  LDAX  D                Get POROM,POR1M
238 EF19 A1                  ANA   C                Only reqd volume
239 EF1A B0                  ORA   B                Update it
240 EF1B 12                  STAX  D                Back in POROM,POR1M
241 EF1C 77                  MOV   M,A              and in PORO,POR1
242 EF1D C1                  POP   B
243 EF1E E1                  POP   H
244 EF1F 0C                  INR   C                SCB count +1
245 EF20 79                  MOV   A,C
246 EF21 FE04                CPI   :04
247 EF23 CAA4EF              JZ    :EFA4            Ready if block was NCB
248
```

```
250
251 EF26 23                 INX   H
252 EF27 7E                 MOV   A,M       Get glissando flag
253 EF28 3D                 DCR   A
254 EF29 FA8BEF             JM    :EF8B     Ready if end period
255                                           reached
256 EF2C 23                 INX   H
257 EF2D 5E                 MOV   E,M       ) Get current period of
258 EF2E 23                 INX   H         ) output in DE
259 EF2F 56                 MOV   D,M       )
260 EF30 E5                 PUSH  H
261 EF31 23                 INX   H
262 EF32 7E                 MOV   A,M       ) Get reqd final period
263 EF33 23                 INX   H         ) in HL
264 EF34 66                 MOV   H,M       )
265 EF35 6F                 MOV   L,A
266 EF36 C23BEF             JNZ   :EF3B     Jump if end period not
267                                           reached
268 EF39 54                 MOV   D,H       ) HL=DE if 'set freq'
269 EF3A 5D                 MOV   E,L       )
270 EF3B CD14DE     L1E260  CALL  :DE14     Compare HL-DE
271 EF3E F5                 PUSH  PSW
272 EF3F D5                 PUSH  D
273 EF40 D244EF             JNC   :EF44     Jump if final period >=
274                                           current period
275 EF43 EB                 XCHG            Else exchange values
276 EF44 CD1ADE     L1E261  CALL  :DE1A     Calc difference in HL
277 EF47 D1                 POP   D
278 EF48 D5                 PUSH  D
279 EF49 E5                 PUSH  H
280 EF4A EB                 XCHG
281 EF4B 1E40               MVI   E,:40     )
282 EF4D 7D         L1E262  MOV   A,L       )
283 EF4E 17                 RAL             )
284 EF4F 6F                 MOV   L,A       )
285 EF50 7C                 MOV   A,H       )
286 EF51 17                 RAL             ) HL = HL SHL 6
287 EF52 67                 MOV   H,A       )
288 EF53 7B                 MOV   A,E       )
289 EF54 17                 RAL             )
290 EF55 5F                 MOV   E,A       )
291 EF56 D24DEF             JNC   :EF4D
292 EF59 6C                 MOV   L,H       ) HL = 1/64 orig. value
293 EF5A 63                 MOV   H,E       )
294 EF5B 7C                 MOV   A,H
295 EF5C B5                 ORA   L
296 EF5D C261EF             JNZ   :EF61
297 EF60 23                 INX   H
298 EF61 D1         L1E263  POP   D
299 EF62 CD14DE             CALL  :DE14     Compare HL-DE
300 EF65 0602               MVI   B,:02
301 EF67 DA6DEF             JC    :EF6D     Jump if DE > HL
302 EF6A 0600               MVI   B,:00
303 EF6C EB                 XCHG
304 EF6D D1         L1E264  POP   D
305 EF6E F1                 POP   PSW
306 EF6F D275EF             JNC   :EF75
307 EF72 CD26DE             CALL  :DE26     HL is its 2-compl.
308 EF75 19         L1E265  DAD   D
309 EF76 EB                 XCHG
310 EF77 E1                 POP   H
311 EF78 72                 MOV   M,D       ) Set new current period
```

```
312 EF79 2B                 DCX   H          )
313 EF7A 73                 MOV   M,E        )
314 EF7B 2B                 DCX   H
315 EF7C 70                 MOV   M,B        Set glissando flag
316 EF7D C5                 PUSH  B
317 EF7E 79                 MOV   A,C        )
318 EF7F 3D                 DCR   A          ) Calc offset for oscill.
319 EF80 87                 ADD   A          ) address
320 EF81 4F                 MOV   C,A        )
321 EF82 0600               MVI   B,:00
322 EF84 2100FC             LXI   H,:FC00    Addr osc. channel 0
323 EF87 09                 DAD   B          HL = addr current osc
324 EF88 73                 MOV   M,E        ) Load oscillator
325 EF89 72                 MOV   M,D        )
326 EF8A C1                 POP   B
327
328                 * Block done:
329
330 EF8B 110E00     L1E266  LXI   D,:000E
331 EF8E E1                 POP   H          Get startadr prev. block
332 EF8F 19                 DAD   D          HL pnts to next block
333 EF90 C373EE             JMP   :EE73      Run next block
334                 *
335                 * RESTART ENVELOPE:
336                 *
337                 * Gets 1st volume of envelope.
338                 *
339                 * Entry: DE: Points to 2nd byte of envelope pointer
340                 *            of a SCB.
341                 * Exit:  A:  Volume.
342                 *        HL: Address volume field in envelope.
343                 *        BCDEF preserved.
344                 *
345 EF93 D5         L1E267  PUSH  D          Save pntr
346 EF94 13                 INX   D
347 EF95 1A                 LDAX  D          Get lobyte env pntr
348 EF96 6F                 MOV   L,A        in L
349 EF97 13                 INX   D
350 EF98 1A                 LDAX  D          Get hibyte env pntr
351 EF99 67                 MOV   H,A        in H
352 EF9A D1                 POP   D          Restore pntr
353 EF9B 7E                 MOV   A,M        Get volume in A
354 EF9C C9                 RET
355                 *
356                 * IF SOUND OF BLOCK IS 'OFF': GOTO NEXT BLOCK:
357                 *
358 EF9D 0C         L1E268  INR   C          Update SCB count
359 EF9E 79                 MOV   A,C
360 EF9F FE04               CPI   :04        All blocks done ?
361 EFA1 C28BEF             JNZ   :EF8B      Next block if not
362 EFA4 E1         L1E269  POP   H
363 EFA5 C9                 RET
364                 *
365                 ***************
366                 * BANK RETURN *
367                 ***************
368                 *
369                 * Goto ROM bank 0.
370                 *
371                 * Entry: A: Current PROM.
372                 *
373 EFA6 C1         BRET    POP   B
```

```
374 EFA7 E63F              ANI   :3F       Select ROM bank 0
375 EFA9 C308D8            JMP   :D808     Load PORO/POROM
376                 *
377 EFAC FF                DATA  :FF
378 EFAD FF                DATA  :FF
379 EFAE FF                DATA  :FF
380 EFAF FF                DATA  :FF
381 EFB0 FF                DATA  :FF
382 EFB1 FF                DATA  :FF
383 EFB2 FF                DATA  :FF
384 EFB3 FF                DATA  :FF
385 EFB4 FF                DATA  :FF
386                 *
387                 ****************************************
388                 * LOADA: READ VARIABLE TYPE FROM TAPE *
389                 ****************************************
390                 *
391                 * Reads 1 byte from tape.
392                 *
393                 * Exit: A: Variable type.
394                 *       DEHL preserved.
395                 *
396 EFB5 E5          R1BB    PUSH  H
397 EFB6 D5                  PUSH  D
398 EFB7 213E01              LXI   H,:013E   Startaddr EBUF
399 EFBA 113F01              LXI   D,:013F   Next addr
400 EFBD CDD102              CALL  :02D1     Read type from tape
401 EFC0 D2B3D2              JNC   :D2B3     Evt run 'LOADING ERROR ..'
402 EFC3 3A3E01              LDA   :013E     Get var.type in A
403 EFC6 D1                  POP   D
404 EFC7 E1                  POP   H
405 EFC8 C9                  RET
406                 *
407 EFC9 FF                DATA  :FF
408 EFCA FF                DATA  :FF
409 EFCB FF                DATA  :FF
410 EFCC FF                DATA  :FF
411 EFCD FF                DATA  :FF
412 EFCE FF                DATA  :FF
413 EFCF FF                DATA  :FF
414 EFD0 FF                DATA  :FF
415 EFD1 FF                DATA  :FF
416 EFD2 FF                DATA  :FF
417 EFD3 FF                DATA  :FF
418 EFD4 FF                DATA  :FF
419 EFD5 FF                DATA  :FF
420 EFD6 FF                DATA  :FF
421 EFD7 FF                DATA  :FF
422 EFD8 FF                DATA  :FF
423 EFD9 FF                DATA  :FF
424 EFDA FF                DATA  :FF
425 EFDB FF                DATA  :FF
426 EFDC FF                DATA  :FF
427 EFDD FF                DATA  :FF
428 EFDE FF                DATA  :FF
429 EFDF FF                DATA  :FF
430 EFE0 FF                DATA  :FF
431 EFE1 FF                DATA  :FF
432 EFE2 FF                DATA  :FF
433 EFE3 FF                DATA  :FF
434 EFE4 FF                DATA  :FF
435 EFE5 FF                DATA  :FF
```

```
436 EFE6 FF                    DATA  :FF
437 EFE7 FF                    DATA  :FF
438 EFE8 FF                    DATA  :FF
439 EFE9 FF                    DATA  :FF
440 EFEA FF                    DATA  :FF
441 EFEB FF                    DATA  :FF
442 EFEC FF                    DATA  :FF
443 EFED FF                    DATA  :FF
444 EFEE FF                    DATA  :FF
445 EFEF FF                    DATA  :FF
446 EFF0 FF                    DATA  :FF
447 EFF1 FF                    DATA  :FF
448 EFF2 FF                    DATA  :FF
449 EFF3 FF                    DATA  :FF
450 EFF4 FF                    DATA  :FF
451 EFF5 FF                    DATA  :FF
452 EFF6 FF                    DATA  :FF
453 EFF7 FF                    DATA  :FF
454 EFF8 FF                    DATA  :FF
455                    *
456                    ***************************
457                    * part of 'EXP' (1E667) *
458                    ***************************
459                    *
460 EFF9 B4            L1E272  ORA   H
461 EFFA F2B8E6                JP    :E6B8
462 EFFD C3ADE6                JMP   :E6AD
463                    *
464                    *
465                    *
466 F000                       END
```

```
*****************************
* S Y M B O L   T A B L E *
*****************************
```

```
BRET   EFA6   L1E245 EE0B   L1E246 EE2F   L1E247 EE38
L1E248 EE4C   L1E250 EE73   L1E251 EE7E   L1E252 EEA9
L1E253 EEB1   L1E254 EEB8   L1E255 EEBD   L1E256 EEDC
L1E257 EEEB   L1E258 EF12   L1E259 EF18   L1E260 EF3B
L1E261 EF44   L1E262 EF4D   L1E263 EF61   L1E264 EF6D
L1E265 EF75   L1E266 EF8B   L1E267 EF93   L1E268 EF9D
L1E269 EFA4   L1E272 EFF9   L1E273 EE0F   R1BB   EFB5
TEMP0  EE6E
```

```
002                             ORG   :E000
003                     *
004                     *
005                     *
006                     *   ======================
007                     *** SCREEN DRIVING PACKAGE ***
008                     *   ======================
009                     *
010                     * Called by RST 5; DATA XX. XX indicates the
011                     * offset of E000 for the different entrypoints.
012                     *
013                     ***************
014                     * ENTRYPOINTS *
015                     ***************
016                     *
017                     * Screen functions:
018
019 E000 C3C3E0         ZSINIT  JMP   :E0C3     Initialise screen
020 E003 C302E1         ZSOUTC  JMP   :E102     Output one character
021 E006 C337E2         ZSCLT   JMP   :E237     Set text colours
022 E009 C379E2         ZSCUS   JMP   :E279     Set cursor position
023 E00C C3CCE2         ZSCUA   JMP   :E2CC     Ask cursor position and
024                                               size character screen
025 E00F C316E3         ZSCUM   JMP   :E316     Set cursor mode
026 E012 C344E3         ZSCUI   JMP   :E344     Flash cursor
027 E015 C38BE3         ZSFETC  JMP   :E38B     Get character from line
028 E018 C3D9E3         ZSMODE  JMP   :E3D9     Change mode
029 E01B C3A4E6         ZSCLG   JMP   :E6A4     Set graphics colours
030 E01E C310E7         ZSDOT   JMP   :E710     Draw a dot on the screen
031 E021 C31BE7         ZSDRAW  JMP   :E71B     Draw a line on the screen
032 E024 C318E8         ZSFILL  JMP   :E818     Fill a rectangular area
033 E027 C384E8         ZSCRN   JMP   :E884     Ask colour of a point on the
034                                               screen and the size of the
035                                               graphics screen
036                     * Edit functions:
037
038 E02A C3F4EB         ZEDIT   JMP   :EBF4     Initialise editor
039 E02D C31EEC         ZEDOB   JMP   :EC1E     Run edit command
040                     *
041                     ************************
042                     * CONSTANT TABLE MODE 0 *
043                     ************************
044                     *
045                     * These constant tables are moved into the screen
046                     * variables in RAM (0084-0098) when the appropriate
047                     * mode is entered.
048                     *
049                     * Except the last 4 data blocks, all values
050                     * are offset from the screen top address (BFFF for
051                     * a 48K machine). This is valid for all modes.
052                     *
053 E030 B00C         CON0    DBL   :0CB0     First free RAM byte
054 E032 0000                 DBL   :0000     Top of rolled area
055 E034 0000                 DBL   :0000     End graphics area
056 E036 1000                 DBL   :0010     Start character area
057 E038 A00C                 DBL   :0CA0     End character area
058 E03A B00C                 DBL   :0CB0     End screen
059 E03C 0000                 DBL   :0000     End area used splitting mode
060 E03E 0000                 DBL   :0000     Start archive save area
061                     *
062 E040 0000                 DBL   :0000     Number of blobs horizontally
063 E042 00                   DATA  :00       Number of lines of graphics
```

```
064 E043 00                 DATA  :00       Number saved graphics lines
065 E044 00                 DATA  :00       Number of bytes/line
066                 *
067                 ****************************
068                 * CONSTANT TABLE MODES 1/2 *
069                 ****************************
070                 *
071 E045 3806       CON1    DBL   :0638     First free RAM byte
072 E047 3001               DBL   :0130     Top area rolled up for mode
073 E049 2806               DBL   :0628     End graphics area
074 E04B 2806               DBL   :0628     CHS (dummy)
075 E04D 6008               DBL   :0860     End character area
076 E04F 3806               DBL   :0638     End screen
077 E051 4807               DBL   :0748     End area used splitting mode
078 E053 4007               DBL   :0740     Start graphic archive area
079                 *
080 E055 4800               DBL   :0048     Number of blobs horizontally
081 E057 41                 DATA  :41       Number of lines of graphics
082 E058 0C                 DATA  :0C       Number archive area lines
083 E059 18                 DATA  :18       Number of bytes/line
084                 *
085                 ******************************
086                 * CONSTANT TABLE MODES 1A/2A *
087                 ******************************
088                 *
089 E05A 6008       CON1A   DBL   :0860     First free RAM byte
090 E05C 3001               DBL   :0130     Top of rolled area
091 E05E 0805               DBL   :0508     End graphics area
092 E060 1805               DBL   :0518     Start character area
093 E062 3007               DBL   :0730     End character area
094 E064 4007               DBL   :0740     End screen
095 E066 4807               DBL   :0748     End area used splitting mode
096 E068 2806               DBL   :0628     Start graph temp save area
097                 *
098 E06A 4800               DBL   :0048     Number of blobs horizontally
099 E06C 41                 DATA  :41       Number of lines of graphics
100 E06D 0C                 DATA  :0C       Number saved graphics lines
101 E06E 18                 DATA  :18       Number of bytes/line
102                 *
103                 ****************************
104                 * CONSTANT TABLE MODES 3/4 *
105                 ****************************
106                 *
107 E06F 7C17       CON3    DBL   :177C     First free RAM byte
108 E071 6004               DBL   :0460     Top area rolled up
109 E073 6C17               DBL   :176C     End graphics area
110 E075 6C17               DBL   :176C     CHS (dummy)
111 E077 A419               DBL   :19A4     End character area
112 E079 7C17               DBL   :177C     End screen
113 E07B BC1B               DBL   :1BBC     End area used splitting mode
114 E07D 5415               DBL   :1554     Start graph archive area
115                 *
116 E07F A000               DBL   :00A0     Number of blobs horizontally
117 E081 82                 DATA  :82       Number of lines of graphics
118 E082 18                 DATA  :18       Number archive area lines
119 E083 2E                 DATA  :2E       Number of bytes/line
120                 *
121                 ******************************
122                 * CONSTANT TABLE MODES 3A/4A *
123                 ******************************
124                 *
125 E084 A419       CON3A   DBL   :19A4     First free RAM byte
```

```
126 E086 6004                   DBL   :0460     Top of rolled area
127 E088 1C13                   DBL   :131C     End graphics area
128 E08A 2C13                   DBL   :132C     Start character area
129 E08C 4415                   DBL   :1544     End character area
130 E08E 5415                   DBL   :1554     End screen
131 E090 BC1B                   DBL   :1BBC     End area used splitting mode
132 E092 6C17                   DBL   :176C     Start graph temp save area
133                     *
134 E094 A000                   DBL   :00A0     Number of blobs horizontally
135 E096 82                     DATA  :82       Number of lines of graphics
136 E097 18                     DATA  :18       Number saved graphics lines
137 E098 2E                     DATA  :2E       Number of bytes/line
138                     *
139                     ******************************
140                     * CONSTANT TABLE MODES 5/6 *
141                     ******************************
142                     *
143 E099 205A           CON5    DBL   :5A20     First free RAM byte
144 E09B 880F                   DBL   :0F88     Top area rolled up
145 E09D 105A                   DBL   :5A10     End graphics area
146 E09F 105A                   DBL   :5A10     CHS (dummy)
147 E0A1 485C                   DBL   :5C48     End character area
148 E0A3 205A                   DBL   :5A20     End screen
149 E0A5 8869                   DBL   :6988     End area used splitting mode
150 E0A7 D04C                   DBL   :4CD0     Start graph archive area
151                     *
152 E0A9 5001                   DBL   :0150     Number of blobs horizontally
153 E0AB 00                     DATA  :00       Number of lines of graphics
154 E0AC 2C                     DATA  :2C       Number saved graphics lines
155 E0AD 5A                     DATA  :5A       Number of bytes/line
156                     *
157                     ********************************
158                     * CONSTANT TABLE MODES 5A/6A *
159                     ********************************
160                     *
161 E0AE 485C           CON5A   DBL   :5C48     First free RAM byte
162 E0B0 880F                   DBL   :0F88     Top of rolled area
163 E0B2 984A                   DBL   :4A98     End graphics area
164 E0B4 A84A                   DBL   :4AA8     Start character area
165 E0B6 C04C                   DBL   :4CC0     End character area
166 E0B8 D04C                   DBL   :4CD0     End screen
167 E0BA 8869                   DBL   :6988     End area used splitting mode
168 E0BC 105A                   DBL   :5A10     Start graph temp save area
169                     *
170 E0BE 5001                   DBL   :0150     Number of blobs horizontally
171 E0C0 00                     DATA  :00       Number of lines of graphics
172 E0C1 2C                     DATA  :2C       Number saved graphics lines
173 E0C2 5A                     DATA  :5A       Number of bytes/line
174                     *
175                     **********************
176                     * INITIALISE SCREEN *
177                     **********************
178                     *
179                     * The screen is initialised into all character
180                     * format (mode 0), and the cursor mode is set
181                     * and it is positioned in the top left corner.
182                     * The normal mode set routine is used (memory
183                     * management routine).
184                     *
185                     * This is the only time the package is told the
186                     * startaddress of the screen. The colour format
                        * is as for SCOLT, SCOLG. The cursor format is
```

```
188                 * as for SCURM.
189                 *
190                 * Entry: HL: Top location screen RAM.
191                 *        DE: Points to list with initialisation
192                 *            parameters (start at C7E0).
193                 * Exit:  All registers maybe corrupted.
194                 *
195 E0C3 228000      SINIT   SHLD  :0080      Store startaddr screen
196 E0C6 D5                 PUSH  D
197 E0C7 11F0FF             LXI   D,:FFF0
198 E0CA 19                 DAD   D
199 E0CB 228200             SHLD  :0082      Set top of screen
200 E0CE E1                 POP   H
201 E0CF AF                 XRA   A
202 E0D0 329D00             STA   :009D      Select mode 1
203 E0D3 CD16E3             CALL  :E316      Set cursor type + info
204 E0D6 23                 INX   H
205 E0D7 23                 INX   H
206 E0D8 CD37E2             CALL  :E237      Init. colours COLORT
207 E0DB 3D                 DCR   A
208 E0DC 329D00             STA   :009D      Select mode 0
209 E0DF 110400             LXI   D,:0004
210 E0E2 19                 DAD   D          Get addr COLORG parameters
211 E0E3 CDA4E6             CALL  :E6A4      Init. colours COLORG
212 E0E6 19                 DAD   D          Get addr screen management
213                                          parameters
214 E0E7 5E                 MOV   E,M        )
215 E0E8 23                 INX   H          ) Get addr screen management
216 E0E9 56                 MOV   D,M        ) routine
217 E0EA 23                 INX   H
218 E0EB EB                 XCHG
219 E0EC 22C400             SHLD  :00C4      Store addr SMKRM
220 E0EF EB                 XCHG
221 E0F0 5E                 MOV   E,M        ) Get addr emergency
222 E0F1 23                 INX   H          ) stop routine
223 E0F2 56                 MOV   D,M        )
224 E0F3 EB                 XCHG
225 E0F4 22C600             SHLD  :00C6      Store addr em.stop routine
226 E0F7 3E10               MVI   A,:10
227 E0F9 329D00             STA   :009D      Select init. screen mode
228                                          (no text, no graphics)
229 E0FC 3EFF               MVI   A,:FF
230 E0FE CDD9E3             CALL  :E3D9      Set up screen for mode 0
231 E101 C9                 RET
232                 *
233                 ***********************************
234                 * OUTPUT A CHARACTER TO SCREEN *
235                 ***********************************
236                 *
237                 * Displays one character on the screen.
238                 *
239                 * Entry: A: Character to be displayed.
240                 * Exit:  ABCDEHL preserved.
241                 *        CY=1: Character ignored.
242                 *
243 E102 37          SOUTC   STC              CY=1
244 E103 F5                 PUSH  PSW
245 E104 C5                 PUSH  B
246 E105 D5                 PUSH  D
247 E106 E5                 PUSH  H
248 E107 CD1CE2             CALL  :E21C      Change to char mode if
249                                          not yet done
```

```
250 E10A 2A7200            LHLD  :0072     Get cursor position
251 E10D CD6BE3            CALL  :E36B     Delete cursor
252 E110 FE0D              CPI   :0D       Car.ret ?
253 E112 CA3DE1            JZ    :E13D     Then print it
254 E115 FE0C              CPI   :0C       Form feed ?
255 E117 CA59E1            JZ    :E159     Then clear screen
256 E11A FE08              CPI   :08       Backspace ?
257 E11C CA66E1            JZ    :E166     Then cancell last character
258 E11F F5                PUSH  PSW
259 E120 3A7A00            LDA   :007A     Get addr last byte on line
260 E123 BD                CMP   L         Reached ?
261 E124 CAA9E1            JZ    :E1A9     Then extend lines
262 E127 F1        OTC05   POP   PSW
263 E128 77                MOV   M,A       Put char on screen
264 E129 2B                DCX   H
265 E12A 2B                DCX   H         Points to next screen loc
266 E12B CD30E3    OTC10   CALL  :E330     Put cursor on screen
267                *
268                OTC20
269 E12E E1        XRCC    POP   H
270 E12F D1                POP   D
271 E130 C1                POP   B
272 E131 F1                POP   PSW
273 E132 3F                CMC             CY=0: char accepted
274 E133 C9                RET
275
276                * If character not accepted:
277
278 E134 F1        OTC25   POP   PSW
279 E135 CD30E3    OTC26   CALL  :E330     Put cursor on screen
280 E138 E1        XRET    POP   H
281 E139 D1                POP   D
282 E13A C1                POP   B
283 E13B F1                POP   PSW       CY=1: char ignored
284 E13C C9                RET
285
286                * If carriage return:
287
288 E13D 2A7800    OTC30   LHLD  :0078     Get startaddr current line
289 E140 EB                XCHG            in DE
290 E141 217AFF            LXI   H,:FF7A
291 E144 19                DAD   D         Get startaddr next line
292 E145 EB                XCHG            in DE
293 E146 2A8C00            LHLD  :008C     Get end char area
294 E149 CDFBE6            CALL  :E6FB     Check if end is reached
295 E14C EB                XCHG            Next line mode byte in HL
296 E14D CCCBE1            CZ    :E1CB     If end reached: scroll up
297                                        one line
298 E150 CCFDE1            CZ    :E1FD     and init. this line with
299                                        blanks
300 E153 CD87E6    OTC35   CALL  :E687     Cursor on begin next line
301 E156 C32EE1            JMP   :E12E     Quit; char accepted
302
303                * If form feed:
304
305 E159 2A8C00    OTC40   LHLD  :008C     Get end character area
306 E15C EB                XCHG            in DE
307 E15D 2A8A00            LHLD  :008A     Get start character area
308 E160 CDFDE1            CALL  :E1FD     Init screen with spaces
309 E163 C353E1            JMP   :E153     Cursor top left corner of
310                                        char area; popall; ret
311
```

```
312                     * If backspace:
313
314 E166 EB         OTC50   XCHG            Cursor position in DE
315 E167 2A7B00             LHLD  :007B     Get startaddr current line
316 E16A 01F8FF             LXI   B,:FFF8   Left border width
317 E16D 09                 DAD   B         Get addr 1st char on line
318 E16E CDFBE6             CALL  :E6FB     Cursor at begin of line?
319 E171 EB                 XCHG
320 E172 CA35E1             JZ    :E135     Then ignore char; abort
321 E175 23                 INX   H         ) Cursor one location
322 E176 23                 INX   H         ) backwards
323 E177 3620               MVI   M,:20     Load space in this location
324 E179 3A7B00             LDA   :007B     Get number of extended lines
325 E17C B7                 ORA   A
326 E17D CA2BE1             JZ    :E12B     If no cont line: put cursor
327                                           on screen
328 E180 FA2BE1             JM    :E12B     If char accepted: put
329                                           cursor on screen
330
331                     * Backspace on a continuation line:
332
333 E183 D5                 PUSH  D         Save addr 1st byte on line
334                                           on stack
335 E184 EB                 XCHG            HL is cursor position
336 E185 01F2FF             LXI   B,:FFF2
337 E188 09                 DAD   B         HL = end indent area
338 E189 CDFBE6             CALL  :E6FB     Compare DE-HL
339 E18C EB                 XCHG
340 E18D D1                 POP   D
341 E18E C22BE1             JNZ   :E12B     If not there: put cursor on
342                                           screen; quit, char accepted
343 E191 EB                 XCHG
344 E192 3620               MVI   M,:20     Else cancel cont char (C)
345 E194 217B00             LXI   H,:007B
346 E197 35                 DCR   M         Decr. number extended lines
347 E198 2A7B00             LHLD  :007B     Get startaddr current line
348 E19B 118600             LXI   D,:0086
349 E19E 19                 DAD   D         Pnts to start previous line
350 E19F CD98E6             CALL  :E698     Store addr line mode byte as
351                                           current one and set last
352                                           byte on that line
353 E1A2 1180FF             LXI   D,:FF80
354 E1A5 19                 DAD   D
355 E1A6 C32BE1             JMP   :E12B     Put cursor on screen; quit,
356                                           char accepted
357
358                     * If end of line is reached:
359
360 E1A9 3A7B00     OTC80   LDA   :007B     Get number extended lines
361 E1AC FE03               CPI   :03       Max (3) reached ?
362 E1AE D234E1             JNC   :E134     Then put cursor on screen,
363                                           ret
364 E1B1 3C                 INR   A         Incr. number ext. lines
365 E1B2 47                 MOV   B,A       Store it in B
366 E1B3 3E0D               MVI   A,:0D
367 E1B5 CD02E1             CALL  :E102     Output car.ret
368 E1B8 78                 MOV   A,B
369 E1B9 327B00             STA   :007B     Update nr ext. lines
370 E1BC 2A7200             LHLD  :0072     Get cursor position addr
371 E1BF CD6BE3             CALL  :E36B     Delete cursor
372 E1C2 3643               MVI   M,:43     Print 'C' at left of line
373 E1C4 11F2FF             LXI   D,:FFF2
```

```
374 E1C7 19                 DAD   D          Indent 6 pos
375 E1C8 C327E1             JMP   :E127      Store char on new pos; put
376                                          cursor on screen
377                 *
378                 *************
379                 * SCROLLING *
380                 *************
381                 *
382                 * Scrolls up text area. Moves the character area of
383                 * the screen up one line.
384                 * Only the characters are moved, not the control and
385                 * colour bytes.
386                 *
387                 * Entry: None.
388                 * Exit:  AF preserved, BC corrupted.
389                 *        DE: End of bottom line.
390                 *        HL: Start of bottom line.
391                 *
392 E1CB 017AFF     SCROLL LXI   B,:FF7A   -86 (length one line)
393
394                 * Entry from Edit:
395                 * Scroll screen for number of positions given in
396                 * BC (-2 = 1 position left):
397
398 E1CE F5         SCR10  PUSH  PSW
399 E1CF 2A8A00            LHLD  :008A      Get startaddr character area
400 E1D2 54                MOV   D,H        ) and store it in DE
401 E1D3 5D                MOV   E,L        )
402 E1D4 09                DAD   B          Get addr line mode byte
403                                          next line
404 E1D5 EB                XCHG             in DE
405 E1D6 01F8FF     SCR20  LXI   B,:FFF8
406 E1D9 09                DAD   B          Get 1st useable location
407                                          on 1st line
408 E1DA EB                XCHG             in DE
409 E1DB 09                DAD   B          Get 1st useable location
410                                          on 2nd line
411 E1DC EB                XCHG             in DE; 1st line in HL
412 E1DD 063C              MVI   B,:3C      max 60 characters
413 E1DF 1A         SCR30  LDAX  D          Get char from 2nd line
414 E1E0 77                MOV   M,A        and move it to 1st line
415 E1E1 1B                DCX   D
416 E1E2 1B                DCX   D          Next char 2nd line
417 E1E3 2B                DCX   H
418 E1E4 2B                DCX   H          Next loc 1st line
419 E1E5 05                DCR   B
420 E1E6 C2DFE1            JNZ   :E1DF      Next char to be moved 1 line
421 E1E9 01FAFF            LXI   B,:FFFA
422 E1EC 09                DAD   B          Get addr line mode byte
423                                          2nd line
424 E1ED EB                XCHG             in DE
425 E1EE 09                DAD   B          Get addr line mode byte
426                                          3rd line
427 E1EF EB                XCHG             in DE; 2nd line in HL
428 E1F0 E5                PUSH  H
429 E1F1 2A8C00            LHLD  :008C      Get addr end character area
430 E1F4 CDFBE6            CALL  :E6FB      Check if end reached
431 E1F7 E1                POP   H
432 E1F8 DAD6E1            JC    :E1D6      If not at end: scroll next
433                                          line
434 E1FB F1                POP   PSW
                           RET
```

```
436                     *
437                     *
438                     *
439 E1FD                        END
```

```
****************************
* S Y M B O L   T A B L E *
****************************
```

CON0	E030	CON1	E045	CON1A	E05A	CON3	E06F
CON3A	E084	CON5	E099	CON5A	E0AE	OTC05	E127
OTC10	E12B	OTC20	E12E	OTC25	E134	OTC26	E135
OTC30	E13D	OTC35	E153	OTC40	E159	OTC50	E166
OTC80	E1A9	SCR10	E1CE	SCR20	E1D6	SCR30	E1DF
SCROLL	E1CB	SINIT	E0C3	SOUTC	E102	XRCC	E12E
XRET	E138	ZEDIT	E02A	ZEDOB	E02D	ZSCLG	E01B
ZSCLT	E006	ZSCRN	E027	ZSCUA	E00C	ZSCUI	E012
ZSCUM	E00F	ZSCUS	E009	ZSDOT	E01E	ZSDRAW	E021
ZSFETC	E015	ZSFILL	E024	ZSINIT	E000	ZSMODE	E018
ZSOUTC	E003						

```
002                             ORG   :E1FD
003                     *
004                     *
005                     *
006                     ************************************
007                     * INITIALISE SCREEN CHARACTER AREA *
008                     ************************************
009                     *
010                     * Fills screen with spaces (clears screen).
011                     * The line mode byte is set #7A, the line colour
012                     * byte to #40, all colour bytes to #00 (4-colour
013                     * text), all character bytes to #20.
014                     *
015                     * Entry: HL: 1st byte after header.
016                     *        DE: end character area.
017                     * Exit:  All registers preserved.
018                     *
019 E1FD F5             FILLS   PUSH  PSW
020 E1FE C5                     PUSH  B
021 E1FF D5                     PUSH  D
022 E200 E5                     PUSH  H
023 E201 367A           FIS10   MVI   M,:7A     Set control byte for char
024                                             mode
025 E203 2B                     DCX   H
026 E204 3640                   MVI   M,:40     Set line colour byte
027 E206 2B                     DCX   H
028 E207 0642           FIS20   MVI   B,:42     Number of bytes/line
029 E209 3620           FIS30   MVI   M,:20     Data byte is space
030 E20B 2B                     DCX   H
031 E20C 3600                   MVI   M,:00     Colour byte is 00
032 E20E 2B                     DCX   H
033 E20F 05                     DCR   B
034 E210 C209E2                 JNZ   :E209     Next screen addr
035 E213 CDFBE6                 CALL  :E6FB     All lines done ?
036 E216 C201E2                 JNZ   :E201     Next line if not
037 E219 C338E1                 JMP   :E138     Popall, ret
038                     *
039                     ******************************
040                     * CHANGE TO CHARACTER MODE *
041                     ******************************
042                     *
043                     * If a character is output when the screen is in
044                     * all-graphic mode, the mode is changed to the
045                     * corresponding split-mode.
046                     * If not sufficient space available, mode 0 is
047                     * tried. If still insufficient space, the emergenc
048                     * stop routine is used.
049                     *
050 E21C F5             TMODE   PUSH  PSW
051 E21D 3A9D00                 LDA   :009D     Get current screen mode
052 E220 0F                     RRC             Already character mode ?
053 E221 DA31E2                 JC    :E231     Abort if true
054 E224 37                     STC             CY=1
055 E225 17                     RAL             Set for split mode
056 E226 CDD9E3                 CALL  :E3D9     Change mode
057 E229 3EFF                   MVI   A,:FF
058 E22B DCD9E3                 CC    :E3D9     Change to mode 0 if not
059                                             sufficient space
060 E22E DA33E2                 JC    :E233     Emergency stop if still
061                                             insufficient space
062 E231 F1             TMD10   POP   PSW
063 E232 C9                     RET
```

```
064
065                     * If no space for A-mode or mode 0:
066
067 E233 2AC600      TMD20   LHLD  :00C6      Get addr emergency stop
068                                            routine
069 E236 E9                 PCHL              Go to this routine
070                     *
071                     **********************
072                     * SET TEXT COLOURS *
073                     **********************
074                     *
075                     * The COLORT parameters are set; the header
076                     * and trailer of the character area are
077                     * initialised.
078                     * The colour values are between 0 and F. The
079                     * top 4 bits are ignored.
080                     * The colour change is immediate.
081                     *
082                     * The 1st 2 colours are the default background
083                     * and foreground colours for characters. The last
084                     * 2 are alternative, and may be used for (e.g.)
085                     * the cursor. Colours may be repeated.
086                     *
087                     * Entry: HL points to a vector of 4 bytes containing
088                     *        the colours to be set.
089                     * Exit:  All registers preserved.
090                     *
091 E237 F5           SCOLT   PUSH  PSW
092 E238 C5                   PUSH  B
093 E239 D5                   PUSH  D
094 E23A E5                   PUSH  H
095 E23B 117C00               LXI   D,:007C     Addr 1st byte colour
096                                              register memory
097 E23E CD54E2               CALL  :E254       Init. COLORT reg memory
098 E241 3A9D00               LDA   :009D       Get current screen mode
099 E244 1F                   RAR               Char mode?
100 E245 2A8A00               LHLD  :008A       Get startaddr char area
101 E248 DC67E2               CC    :E267       If char mode: set colours
102                                              header area
103 E24B 2A8E00               LHLD  :008E       Get addr end of screen
104 E24E DC67E2               CC    :E267       If char mode: set colours
105                                              trailer area
106 E251 C338E1               JMP   :E138       Popall, ret
107                     *
108                     ************************
109                     * SET COLOUR PARAMETERS *
110                     ************************
111                     *
112                     * Loads colour data from ROM into the RAM pointers.
113                     * The high nibbles are 8x, 9x, Ax, Bx.
114                     * Used for both COLORT and COLORG.
115                     *
116                     * Entry: HL: Points to colour parameters.
117                     *        DE: Address colour memory in RAM.
118                     * Exit:  DE: Points after colour memory.
119                     *        Other registers corrupted.
120                     *
121 E254 018010      VCOPY   LXI   B,:1080
122 E257 7E          VCP10   MOV   A,M         Get colour from ROM
123 E258 E60F               ANI   :0F
124 E25A B1                 ORA   C           Add bits 4-7
125 E25B 12                 STAX  D           Store in RAM
```

```
126 E25C 23                 INX   H          Next colour
127 E25D 13                 INX   D          Next RAM location
128 E25E 79                 MOV   A,C        )
129 E25F 80                 ADD   B          ) Add #10 to C
130 E260 4F                 MOV   C,A        )
131 E261 FEC0               CPI   :C0        Check if finished
132 E263 C257E2             JNZ   :E257      Next one if not
133 E266 C9                 RET
134                 *
135                 ****************************************
136                 * LOAD COLOURS IN HEADER/TRAILER AREA *
137                 ****************************************
138                 *
139                 * Sets blanking area colour bytes according to
140                 * information given.
141                 * The colourbytes for the character area are
142                 * loaded into the screen header and trailer area.
143                 *
144                 * Entry: HL: Points to 1st control byte after
145                 *            blanking area.
146                 *         DE: Points after table with colours
147                 *            in RAM.
148                 * Exit:  AFDE preserved, BCHL corrupted.
149                 *
150 E267 F5        BCOLS  PUSH  PSW
151 E268 D5               PUSH  D
152 E269 010400           LXI   B,:0004    Distance between colour byte
153 E26C 2B               DCX   H          Addr 1st colour byte of
154                                        screen RAM
155 E26D 1B        BCS10  DCX   D          Addr colour table
156 E26E 1A               LDAX  D          Get colour byte
157 E26F 09               DAD   B          HL = addr in screen RAM
158 E270 77               MOV   M,A        Load byte into screen RAM
159 E271 E630             ANI   :30        Finished ?
160 E273 C26DE2           JNZ   :E26D      Next colourbyte if not
161 E276 D1               POP   D
162 E277 F1               POP   PSW
163 E278 C9               RET
164                 *
165                 ************************
166                 * SET CURSOR POSITION *
167                 ************************
168                 *
169                 * Moves the cursor from its current position to
170                 * any requested position.
171                 * Position 0,0 is the bottom left corner.
172                 *
173                 * Entry: HL contains the y,x position required
174                 *           for the cursor.
175                 * Exit:  BCDEHL preserved.
176                 *        CY=0: OK. F corrupted, A preserved.
177                 *        CY=1: Request off screen.
178                 *              A=01 (error code 'off screen').
179                 *
180 E279 B7        SCURS  ORA   A
181 E27A E5               PUSH  H
182 E27B D5               PUSH  D
183 E27C C5               PUSH  B
184 E27D F5               PUSH  PSW
185 E27E 7D               MOV   A,L        X-coord in A
186 E27F FE3C             CPI   :3C        After end of line ?
                          JNC   :E2C5      Then request off screen
```

```
188 E284 87                     ADD   A          X-coord *2
189 E285 4F                     MOV   C,A        in C
190 E286 0618                   MVI   B,:18      Nr of lines in mode 0
191 E288 3A9D00                 LDA   :009D      Get current screen mode
192 E28B B7                     ORA   A
193 E28C FA95E2                 JM    :E295      Jump if mode 0
194 E28F 0604                   MVI   B,:04      Nr of lines in A-modes
195 E291 1F                     RAR
196 E292 D2C5E2                 JNC   :E2C5      Error if all-graphics mode
197 E295 7C             SCS10   MOV   A,H        Y-coord in A
198 E296 B8                     CMP   B          More than max value
199 E297 D2C5E2                 JNC   :E2C5      Then request off screen
200 E29A CD6BE3                 CALL  :E36B      Delete old cursor
201 E29D 3C                     INR   A
202 E29E 218600                 LXI   H,:0086    Length 1 char line
203 E2A1 CD46EB                 CALL  :EB46      Calc length reqd number of
204                                              lines (HL=A*HL)
205 E2A4 EB                     XCHG             in DE
206 E2A5 2A8C00                 LHLD  :008C      Store end archive area
207 E2A8 19                     DAD   D          Start of reqd line
208 E2A9 CD98E6                 CALL  :E698      Store addr line mode byte
209                                              current line and store last
210                                              byte on that line
211 E2AC 110B00                 LXI   D,:000B
212 E2AF CDF2E6                 CALL  :E6F2      HL=start of right border
213 E2B2 59                     MOV   E,C
214 E2B3 1600                   MVI   D,:00
215 E2B5 CDF2E6                 CALL  :E6F2      Subtract char offset
216 E2B8 CD30E3                 CALL  :E330      Put cursor on screen
217 E2BB 3E00                   MVI   A,:00
218 E2BD 327B00                 STA   :007B      No extended lines
219 E2C0 F1                     POP   PSW        No-error return
220 E2C1 C1             SCS20   POP   B
221 E2C2 D1                     POP   D
222 E2C3 E1                     POP   H
223 E2C4 C9                     RET
224
225                     * If error 'off screen':
226
227 E2C5 F1             SCS30   POP   PSW
228 E2C6 3E01                   MVI   A,:01      Set error code
229 E2C8 3F                     CMC              Change CY to 1
230 E2C9 C3C1E2                 JMP   :E2C1      Pop, ret
231                     *
232                     ***************************************************
233                     * ASK CURSOR POSITION AND SIZE CHARACTER SCREEN *
234                     ***************************************************
235                     *
236                     * Returns the position of the cursor and the
237                     * range of possible values.
238                     * Values given in DE are maximum values of
239                     * coordinates.
240                     * If the mode is all graphics: DE=HL=0.
241                     *
242                     * Entry: None.
243                     * Exit:  HL gives y,x cursor position.
244                     *        DE gives y,x size of character part of
245                     *           the screen (mode 0: #17,#3B; A-modes:
246                     *           #03,#3B).
247                     *        AFBC preserved.
248                     *
249 E2CC F5             SCURA   PUSH  PSW
```

```
250 E2CD C5                 PUSH  B
251 E2CE 210000             LXI   H,:0000
252 E2D1 54                 MOV   D,H
253 E2D2 5D                 MOV   E,L       DE=HL=0
254 E2D3 3A9D00             LDA   :009D     Get current screen mode
255 E2D6 1F                 RAR             Char mode ?
256 E2D7 D213E3             JNC   :E313     Abort if not
257 E2DA 2A7800             LHLD  :0078     Get startaddr cursor line
258 E2DD E5                 PUSH  H         Save it on stack
259 E2DE 11F8FF             LXI   D,:FFF8   Size left border
260 E2E1 19                 DAD   D         Get addr 1st char byte
261 E2E2 EB                 XCHG            in DE
262 E2E3 2A7200             LHLD  :0072     Get cursor pos addr
263 E2E6 EB                 XCHG
264 E2E7 CDF2E6             CALL  :E6F2     Calc difference of cursor
265                                         pos from begin of line
266 E2EA D1                 POP   D         Get startaddr current line
267 E2EB 7D                 MOV   A,L       (x-coord cursor)*2 in A
268 E2EC B7                 ORA   A
269 E2ED 1F                 RAR             Now x-coord cursor in A
270 E2EE F5                 PUSH  PSW       Save it on stack
271 E2EF 2A8C00             LHLD  :008C     Get addr end char area
272 E2F2 018600             LXI   B,:0086   Length 1 char line
273 E2F5 AF                 XRA   A         Init Y-pos
274 E2F6 F5         SCA10   PUSH  PSW       Save it on stack
275 E2F7 09                 DAD   B         Get line mode byte next line
276 E2F8 CDFBE6             CALL  :E6FB     Is current line this line?
277 E2FB CA03E3             JZ    :E303     Then jump
278 E2FE F1                 POP   PSW       Get Y-coord
279 E2FF 3C                 INR   A         Incr it
280 E300 C3F6E2             JMP   :E2F6     Check if on next line
281 E303 E1         SCA20   POP   H         Y-coord cursor in H
282 E304 F1                 POP   PSW       X-coord cursor in A
283 E305 6F                 MOV   L,A       and now in L
284 E306 1617               MVI   D,:17     Nr of lines for mode 0 -1
285 E308 3A9D00             LDA   :009D     Get current screen mode
286 E30B B7                 ORA   A
287 E30C FA11E3             JM    :E311     Jump if mode 0
288 E30F 1603               MVI   D,:03     Nr of lines for A-modes -1
289 E311 1E3B       SCA30   MVI   E,:3B     Nr of char/line -1
290 E313 C1         SCA40   POP   B
291 E314 F1                 POP   PSW
292 E315 C9                 RET
293                 *
294                 *
295                 *
296 E316                    END
```

```
*****************************
* S Y M B O L   T A B L E *
*****************************
```

BCOLS	E267	BCS10	E26D	FILLS	E1FD	FIS10	E201
FIS20	E207	FIS30	E209	SCA10	E2F6	SCA20	E303
SCA30	E311	SCA40	E313	SCOLT	E237	SCS10	E295
SCS20	E2C1	SCS30	E2C5	SCURA	E2CC	SCURS	E279
TMD10	E231	TMD20	E233	TMODE	E21C	VCOPY	E254
VCP10	E257						

```
002                             ORG   :E316
003                     *
004                     *
005                     *
006                     ********************
007                     * SET CURSOR MODE *
008                     ********************
009                     *
010                     * The format of cursor info is 1 byte cursor type
011                     * and 1 byte of information.
012                     * If the type = 0, the cursor flashes in colour.
013                     * The info is a mask which is exored with the
014                     * colour byte for that character to flash it.
015                     * If the type = 1, the cursor alternates between
016                     * the actual character and the one in the info.
017                     *
018                     * If flash entry is never called, cursor will be
019                     * steady in the alternate colour (type 0) or per-
020                     * manently the alternate character (type 1).
021                     *
022                     * Entry: HL points to new cursor info.
023                     * Exit:  All registers preserved.
024                     *
025 E316 F5             SCURM   PUSH  PSW
026 E317 C5                     PUSH  B
027 E318 D5                     PUSH  D
028 E319 E5                     PUSH  H
029 E31A 3A9D00                 LDA   :009D     Get current screen mode
030 E31D 1F                     RAR             Char mode ?
031 E31E DC6BE3                 CC    :E36B     Then delete current cursor
032 E321 7E                     MOV   A,M       Get new cursor type
033 E322 327400                 STA   :0074     Store it in pointer
034 E325 23                     INX   H
035 E326 7E                     MOV   A,M       Get new cursor info
036 E327 327500                 STA   :0075     Store it in pointer
037
038                     * Entry from CURSET:
039
040 E32A DC44E3         SCM10   CC    :E344     Flash cursor once if in
041                                             char mode
042 E32D C338E1                 JMP   :E138     Popall, ret
043                     *
044                     **************
045                     * SET CURSOR *
046                     **************
047                     *
048                     * Sets some cursor on the screen. Does not delete
049                     * a previous cursor. The screen must already be
050                     * in a character mode.
051                     * Gets the contents of the cursor position address
052                     * and stores it in the pointers.
053                     *
054                     * Entry: HL: Address new cursor position.
055                     * Exit:  All registers preserved.
056                     *
057 E330 F5             CURSET  PUSH  PSW
058 E331 C5                     PUSH  B
059 E332 D5                     PUSH  D
060 E333 E5                     PUSH  H
061 E334 E5                     PUSH  H
062 E335 56                     MOV   D,M       Get contents addr pointed at
063                                             by new cursor
```

```
064 E336 2B                 DCX   H
065 E337 2B                 DCX   H
066 E338 2B                 DCX   H
067 E339 5E                 MOV   E,M       Get colour byte of this addr
068 E33A E1                 POP   H
069 E33B CD8DD6             CALL  :D68D     Store contents and colour
070                                         byte in cursor pointers
071 E33E 00                 NOP
072 E33F 00                 NOP
073 E340 37                 STC             CY=1
074 E341 C32AE3             JMP   :E32A     Flash cursor, popall, ret
075                 *
076                 ****************
077                 * FLASH CURSOR *
078                 ****************
079                 *
080                 * Flashes the cursor once if in char mode,
081                 * otherwise does nothing.
082                 *
083                 * Entry: None.
084                 * Exit:  All registers preserved.
085                 *
086                 SCURI
087 E344 F5        CURFL  PUSH  PSW
088 E345 E5               PUSH  H
089 E346 2A7200           LHLD  :0072     Get cursor pos addr
090 E349 7C               MOV   A,H
091 E34A B5               ORA   L         Check if addr is 0000
092 E34B CA5DE3           JZ    :E35D     Abort if no cursor
093 E34E 3A7400           LDA   :0074     Get cursor type
094 E351 B7               ORA   A         Check type
095 E352 3A7500           LDA   :0075     Get cursor info
096 E355 C260E3           JNZ   :E360     Jump if char type
097
098                 * If 'colour' type:
099
100 E358 2B               DCX   H         )
101 E359 2B               DCX   H         ) Get addr colour byte
102 E35A 2B               DCX   H         )
103 E35B AE               XRA   M         Exor mask with colour byte
104 E35C 77        CFL05  MOV   M,A       And reload colour byte
105 E35D E1        CFL10  POP   H
106 E35E F1               POP   PSW
107 E35F C9               RET
108
109                 * If 'char' type:
110
111 E360 BE         CFL20  CMP   M         Check contents screen loc
112 E361 77        CFL30  MOV   M,A       Move cursor info in loc
113 E362 C25DE3           JNZ   :E35D     Abort if contents screen
114                                         loc is changed now
115 E365 3A7700           LDA   :0077     Else: get contents scrn loc
116 E368 C35CE3           JMP   :E35C     Store it in this loc
117                 *
118                 ****************
119                 * DELETE CURSOR *
120                 ****************
121                 *
122                 * Deletes the current cursor. Loads the address
123                 * pointed at by the cursor with the data stored
124                 * in RAM (0076/77).
125                 * Routine valid for character modes only.
```

```
126                     *
127                     * Entry: None.
128                     * Exit:  All registers preserved.
129                     *
130 E36B F5             CURDEL  PUSH  PSW
131 E36C C5                     PUSH  B
132 E36D D5                     PUSH  D
133 E36E E5                     PUSH  H
134 E36F 2A7600                 LHLD  :0076     Get contents cursor loc
135 E372 EB                     XCHG            in DE
136 E373 2A7200                 LHLD  :0072     Get cursor pos addr
137 E376 E5                     PUSH  H         Save it on stack
138 E377 210000                 LXI   H,:0000
139 E37A 227200                 SHLD  :0072     Move cursor to addr 0000
140 E37D E1                     POP   H         Restore cursor pos addr
141 E37E 7C                     MOV   A,H
142 E37F B5                     ORA   L         Check if addr is 0000
143 E380 CA88E3                 JZ    :E388     Abort if no cursor
144 E383 72                     MOV   M,D       Load data into screen loc
145                                             pointed at by cursor
146 E384 2B                     DCX   H
147 E385 2B                     DCX   H
148 E386 2B                     DCX   H
149 E387 73                     MOV   M,E       Load colourbyte into loc
150                                             pointed at
151 E388 C338E1         CDL10   JMP   :E138     Popall, ret
152                     *
153                     ****************************
154                     * GET CHARACTER FROM LINE *
155                     ****************************
156                     *
157                     * Returns a character from some position on
158                     * the current line.
159                     *
160                     * Entry: C: Line position of required character.
161                     *           (max. legal value = 219).
162                     * Exit:  A: Required character (car.ret if at
163                     *           or past cursor).
164                     *        BCDEHLF preserved.
165                     *
166 E38B C5             SFETC   PUSH  B
167 E38C D5                     PUSH  D
168 E38D E5                     PUSH  H
169 E38E F5                     PUSH  PSW
170 E38F 218600                 LXI   H,:0086   Total nr. of bytes/line
171 E392 3A7B00                 LDA   :007B     Get number extended lines
172 E395 CD46EB                 CALL  :EB46     Calc total nr of bytes
173                                             (HL=A*HL)
174 E398 EB                     XCHG            in DE
175 E399 2A7800                 LHLD  :0078     Get addr line mode byte
176                                             current line
177 E39C 19                     DAD   D         Calc start of line on screen
178 E39D 11EAFF                 LXI   D,:FFEA
179 E3A0 19                     DAD   D         End indent area
180 E3A1 EB                     XCHG            in DE
181 E3A2 3EF9                   MVI   A,:F9     1st bytes on line not
182                                             useable
183 E3A4 81                     ADD   C         Add pos of required char on
184                                             line
185 E3A5 F5                     PUSH  PSW
186 E3A6 0600                   MVI   B,:00
187 E3A8 D2B2E3                 JNC   :E3B2     Jump if in 1st 7 positions
```

```
188 E3AB 05              DCR   B
189 E3AC D635      SFC10 SUI   :35       60 useable positions/line
190 E3AE 04              INR   B         Count nr of extended lines
191 E3AF D2ACE3          JNC   :E3AC     Jump if not on this line
192 E3B2 78        SFC20 MOV   A,B       Nr of extensions in A
193 E3B3 21E4FF          LXI   H,:FFE4   Nr of not used bytes/line
194 E3B6 CD46EB          CALL  :EB46     ) Add-ons for line ends
195 E3B9 19              DAD   D         )
196 E3BA F1              POP   PSW       Restore pos of char on line
197 E3BB 5F              MOV   E,A       into E
198 E3BC 3F              CMC
199 E3BD 9F              SBB   A
200 E3BE 57              MOV   D,A       D=char.count - nr of idents
201 E3BF EB              XCHG
202 E3C0 29              DAD   H         Pos *2 due to colour bytes
203 E3C1 EB              XCHG
204 E3C2 CDF2E6          CALL  :E6F2     Calc pos of reqd char
205 E3C5 EB              XCHG            Addr in DE
206 E3C6 2A7200          LHLD  :0072     Get cursor pos addr
207 E3C9 CDFBE6          CALL  :E6FB     Compare it with addr of char
208 E3CC 3E0D            MVI   A,:0D     Car.ret in A
209 E3CE D2D2E3          JNC   :E3D2     If on or after cursor
210 E3D1 1A              LDAX  D         Get character from line
211 E3D2 67        SFC30 MOV   H,A       Save it temporarily
212 E3D3 F1              POP   PSW       Restore flags
213 E3D4 7C              MOV   A,H       Get character in A
214 E3D5 E1              POP   H
215 E3D6 D1              POP   D
216 E3D7 C1              POP   B
217 E3D8 C9              RET
218                *
219                ***************
220                * CHANGE MODE *
221                ***************
222                *
223                * Change the mode of the screen.
224                *
225                * Entry: A: Code new mode.
226                * Exit:  ABCDEHL preserved.
227                *        CY=0: OK.
228                *        CY=1: Insufficient room for mode.
229                *
230 E3D9 37        SSETM STC             CY=1
231 E3DA F5              PUSH  PSW
232 E3DB C5              PUSH  B
233 E3DC D5              PUSH  D
234 E3DD E5              PUSH  H
235 E3DE FEFF            CPI   :FF       Mode 0 ?
236 E3E0 CC07E4          CZ    :E407     Then set up mode 0 screen
237 E3E3 C43EE4          CNZ   :E43E     Else: Set up screen for
238                                      other modes
239 E3E6 DA04E4          JC    :E404     Jump if no room available
240 E3E9 2A8E00          LHLD  :008E     Get end of screen
241 E3EC D5              PUSH  D
242 E3ED 111000          LXI   D,:0010   Nr of bytes in trailer
243 E3F0 19              DAD   D         Get 1st addr trailer area
244 E3F1 D1              POP   D         Get addr 1st colour
245 E3F2 060F            MVI   B,:0F     Depth of blank
246 E3F4 CDFCE5          CALL  :E5FC     Init trailer area
247 E3F7 2A8400          LHLD  :0084     Get 1st free byte
248 E3FA B7              ORA   A         Set flags on scrn mode byte
249 E3FB CDA6E5          CALL  :E5A6     Perform mem. management
```

```
250 E3FE 329D00             STA   :009D     Store current screen r
251 E401 C32EE1             JMP   :E12E     Popall (CY=0), ret
252
253                 * If error:
254
255 E404 C338E1     STM10   JMP   :E138     Popall (CY=1) ret
256                 *
257                 *
258                 *
259 E407                    END

****************************
* S Y M B O L   T A B L E *
****************************

CDL10  E388   CFL05  E35C   CFL10  E35D   CFL20  E360
CFL30  E361   CURDEL E36B   CURFL  E344   CURSET E330
SCM10  E32A   SCURI  E344   SCURM  E316   SFC10  E3AC
SFC20  E3B2   SFC30  E3D2   SFETC  E38B   SSETM  E3D9
STM10  E404
```

```
002                             ORG   :E407
003                     *
004                     *
005                     *
006                     ****************************
007                     * SET UP SCREEN FOR MODE 0 *
008                     ****************************
009                     *
010                     * Sets up a mode 0 screen, whatever the current mode
011                     * is. If already a character area exists, it is
012                     * moved to the top of the new screen.
013                     *
014                     * Entry: None.
015                     * Exit:  All registers corrupted.
016                     *        CY=0: OK.
017                     *        CY=1: No room.
018                     *
019 E407 37             SSM0    STC             CY=1
020 E408 F5                     PUSH  PSW
021 E409 2130E0                 LXI   H,:E030   Startaddr mode 0 table
022 E40C CD45E5                 CALL  :E545     Load pointers with parameter
023 E40F DA3CE4                 JC    :E43C     If no room available
024 E412 117C00                 LXI   D,:007C   Addr text colour table
025 E415 D5                     PUSH  D         Save addr
026 E416 2A8000                 LHLD  :0080     Get 1st byte screen RAM
027 E419 0606                   MVI   B,:06
028 E41B CDFCE5                 CALL  :E5FC     Set up header
029 E41E 3A9D00                 LDA   :009D     Get old screen mode
030 E421 FE10                   CPI   :10       During initialisation ?
031 E423 CA2DE4                 JZ    :E42D     Then jump
032 E426 1F                     RAR             Split screen ?
033 E427 D22DE4                 JNC   :E42D     Jump if not
034 E42A CD35E6                 CALL  :E635     If split mode: move old
035                                             text, cursor, etc
036 E42D EB             SS010   XCHG            Addr after header in DE
037 E42E 2A8C00                 LHLD  :008C     Get addr end char area
038 E431 EB                     XCHG
039 E432 CDFDE1                 CALL  :E1FD     Blank char area
040 E435 D487E6                 CNC   :E687     Set cursor at begin 1st line
041
042                     * Entry from SSMG:
043
044 E438 D1             SS015   POP   D
045 E439 F1                     POP   PSW
046 E43A 3F                     CMC             CY=0
047 E43B C9                     RET
048
049                     * If no room available:
050                     * Entry from SSMG, SSM, SSMA:
051
052 E43C F1             SS020   POP   PSW       CY=1
053 E43D C9                     RET
054                     *
055                     ************************************
056                     * SET UP SCREEN FOR GRAPHIC MODE *
057                     ************************************
058                     *
059                     * Entry: A:  Screen mode (split if odd).
060                     * Exit:  CY=0: O.K.:
061                     *        Split mode:        DE: Addr text colours.
062                     *        All graphic mode: DE: Addr graph colours.
                                AF preserved. BCHL corrupted.
```

```
064                     *       CY=1: Insufficient room.
065                     *
066 E43E 37             SSMG    STC
067 E43F F5                     PUSH  PSW
068 E440 57                     MOV   D,A       Screen mode in D
069 E441 E601                   ANI   :01       Z=1 if full colour mode
070 E443 7A                     MOV   A,D
071 E444 1F                     RAR             Disable split mode bit in A
072 E445 C4B6E4                 CNZ   :E4B6     If split mode: set up screen
073 E448 CC5FE4                 CZ    :E45F     If full colour mode: idem
074 E44B DA3CE4                 JC    :E43C     Abort if no room
075 E44E F1                     POP   PSW       Get mode code
076 E44F F5                     PUSH  PSW
077 E450 D5                     PUSH  D
078 E451 119E00                 LXI   D,:009E   Addr COLORG table
079 E454 2A8000                 LHLD  :0080     Get addr 1st byte screen RAM
080 E457 0606                   MVI   B,:06     Depth each blanking line
081                                             in header -1
082 E459 CDFCE5                 CALL  :E5FC     Set up header with COLORG
083                                             colours
084 E45C C338E4                 JMP   :E438     Quit, all OK
085                     *
086                     * SET UP A FULL GRAPHIC SCREEN:
087                     *
088                     * Sets up a screen RAM for an all-graphics mode.
089                     *
090                     * Entry: A: Mode code /2.
091                     *        D: Mode code.
092                     * Exit:  CY=0: O.K.:
093                     *              DE points to table graphic colours.
094                     *              AF preserved. BCHL corrupted.
095                     *        CY=1: Insufficient space.
096                     *
097 E45F 37             SSM     STC
098 E460 F5                     PUSH  PSW
099 E461 219AE5                 LXI   H,:E59A   Addr table vectors full
100                                             graphic mode
101 E464 CD39E5                 CALL  :E539     Set up screen mode
102 E467 DA3CE4                 JC    :E43C     Jump if no room
103 E46A 3A9D00                 LDA   :009D     Get current screen mode
104 E46D 92                     SUB   D         ) Check if change split
105 E46E 3D                     DCR   A         ) to all graphics
106 E46F CA00D7                 JZ    :D700     Then check if sufficient
107                                             RAM available and change
108                                             mode
109 E472 CD6BE3                 CALL  :E36B     Delete cursor
110 E475 3A9600                 LDA   :0096     Get nr of graphics lines
111 E478 4F                     MOV   C,A       in C
112 E479 2A8200                 LHLD  :0082     Get addr top graph area
113 E47C CDADE5                 CALL  :E5AD     Blank whole screen
114 E47F 119E00         SSM10   LXI   D,:009E   Addr COLORG table
115 E482 F1                     POP   PSW
116 E483 3F                     CMC             CY=0: O.K.
117 E484 C9                     RET
118
119                     * Change from split to all-graphic mode:
120
121 E485 D20FD7         SSM21   JNC   :D70F     Set up screen mode
122 E488 CD6BE3                 CALL  :E36B     Delete cursor
123 E48B 2A8800                 LHLD  :0088     Get addr temp save area
124 E48E 44                     MOV   B,H       ) in BC
125 E48F 4D                     MOV   C,L       )
```

```
126 E490 C5              PUSH  B         Save it on stack
127 E491 2A9200          LHLD  :0092     Get startaddr archive
128                                      area
129 E494 EB              XCHG            in DE
130 E495 2A8C00          LHLD  :008C     Get addr end archive area
131 E498 CDC2E6          CALL  :E6C2     Move archive area into
132                                      temp save area
133 E49B 2A8600          LHLD  :0086     Get addr top of rolled area
134 E49E 44              MOV   B,H       ) in BC
135 E49F 4D              MOV   C,L       )
136 E4A0 2A8200          LHLD  :0082     Get addr top old graphics
137 E4A3 EB              XCHG            in DE
138 E4A4 2A9900          LHLD  :0099     Get end old screen
139 E4A7 CDC2E6          CALL  :E6C2     Move lower part screen
140                                      downwards
141 E4AA 42              MOV   B,D       ) BC is addr where to put
142 E4AB 4B              MOV   C,E       ) archive area
143 E4AC D1              POP   D         Get startaddr temp save area
144 E4AD 2A9000          LHLD  :0090     Get end temp save area
145 E4B0 CDC2E6          CALL  :E6C2     Move temp save area to
146                                      top of screen
147 E4B3 C37FE4          JMP   :E47F     Quit
148                *
149                * SET UP SCREEN FOR SPLIT MODE:
150                *
151                * Sets up a split screen for a given mode in the
152                * lower RAM.
153                *
154                * Entry: A: Mode code /2.
155                *        D: Mode code.
156                * Exit:  CY=0: O.K.:
157                *              DE: Address text colour table.
158                *              AF preserved, BCHL corrupted.
159                *        CY=1: Insufficient space.
160                *
161 E4B6 37        SSMA  STC
162 E4B7 F5              PUSH  PSW
163 E4B8 21A0E5          LXI   H,:E5A0   Startaddr table vectors
164                                      split modes
165 E4BB CD39E5          CALL  :E539     Set up screen mode
166 E4BE DA3CE4          JC    :E43C     Abort if insufficient space
167 E4C1 3A9D00          LDA   :009D     Get old screen mode
168 E4C4 92              SUB   D         ) Check if change from
169 E4C5 3C              INR   A         ) all-graph to split
170 E4C6 F5              PUSH  PSW       Preserve flags
171 E4C7 D5              PUSH  D         and new mode code
172 E4C8 C2F9E4          JNZ   :E4F9     If not splitting old mode:
173                                      clear graph and moved areas
174 E4CB CD06D7          CALL  :D706     Check suff RAM available;
175                                      prepare full graphic mode
176 E4CE 00              NOP
177 E4CF D20DD7          JNC   :D70D     Set up current mode if
178                                      not O.K.
179 E4D2 2A9200          LHLD  :0092     Get start temp save area
180 E4D5 44              MOV   B,H       ) in BC
181 E4D6 4D              MOV   C,L       )
182 E4D7 2A8200          LHLD  :0082     Get addr after header
183 E4DA EB              XCHG            in DE
184 E4DB 2A8600          LHLD  :0086     Get addr top of screen
185 E4DE CDC2E6          CALL  :E6C2     Move top of screen into
186                                      temp save area
187 E4E1 42              MOV   B,D       ) BC is addr top of screen
```

```
188 E4E2 4B                 MOV   C,E       )
189 E4E3 EB                 XCHG            Addr top rolled up area
190 E4E4 2A9900             LHLD  :0099     Get previous end of graphics
191 E4E7 CDC2E6             CALL  :E6C2     Move lower part of screen
192 E4EA 2A8E00             LHLD  :008E     Get final place for archive
193                                         code
194 E4ED 44                 MOV   B,H       ) in BC
195 E4EE 4D                 MOV   C,L       )
196 E4EF 2A9200             LHLD  :0092     Get addr start temp. save
197                                         area
198 E4F2 EB                 XCHG            in DE
199 E4F3 2A9000             LHLD  :0090     Get addr end split mode
200 E4F6 CDC2E6             CALL  :E6C2     Move temp save area into
201                                         archive area
202 E4F9 2A8A00     SMA10   LHLD  :008A     Get start addr char area
203 E4FC EB                 XCHG            in DE
204 E4FD 2A8C00             LHLD  :008C     Get addr end char area
205 E500 3A9D00             LDA   :009D     Get current screen mode
206 E503 1F                 RAR             Check mode
207 E504 0E04               MVI   C,:04     Nr of char lines in A-mode
208 E506 EB                 XCHG
209 E507 D4FDE1             CNC   :E1FD     Blank char area
210 E50A D487E6             CNC   :E687     Cursor on begin of line
211 E50D DC35E6             CC    :E635     Find old text and move it
212 E510 D1                 POP   D
213 E511 2A8200             LHLD  :0082     Get addr after header
214 E514 3A9700             LDA   :0097     Get nr saved graphics lines
215 E517 4F                 MOV   C,A       in C
216 E518 3A9600             LDA   :0096     Get nr of graphics lines
217 E51B 91                 SUB   C         minus saved ones
218 E51C 4F                 MOV   C,A       stored in C
219 E51D F1                 POP   PSW
220 E51E C4ADE5             CNZ   :E5AD     Blank visible graph area
221 E521 2A8E00             LHLD  :008E     Get addr end of screen
222 E524 3A9700             LDA   :0097     Get nr saved graphics lines
223 E527 4F                 MOV   C,A       in C
224 E528 C4ADE5             CNZ   :E5AD     Blank saved graph area
225 E52B 117C00             LXI   D,:007C   Addr text colour table
226 E52E 0600               MVI   B,:00     Middle as narrow as possible
227 E530 2A8800             LHLD  :0088     Get addr middle area
228 E533 F1                 POP   PSW       Get mode code
229 E534 CDFCE5             CALL  :E5FC     Set up middle area
230                                         (blanking)
231 E537 3F                 CMC
232 E538 C9                 RET
233                 *
234                 * SET UP SCREEN FOR MODE:
235                 *
236                 * Selects the right table according to the mode
237                 * number and sets the screen variables.
238                 *
239                 * Entry: A:  Mode code /2.
240                 *        HL: Points to screen parameter vectors
241                 *            for each pair of modes.
242                 * Exit:  CY=0: O.K.:
243                 *              AFHL corrupted, BCDE preserved.
244                 *        CY=1: Insufficient space.
245                 *
246 E539 E60E          TABP    ANI   :0E       Bits 1,2,3 only
247 E53B CD01E7               CALL  :E701     Add offset to start table
248 E53E 00                   NOP
249 E53F 00                   NOP
```

```
250 E540 00                 NOP
251 E541 7E                 MOV   A,M       )
252 E542 23                 INX   H         ) Get addr from table in HL
253 E543 66                 MOV   H,M       )
254 E544 6F                 MOV   L,A       )
255                 *
256                 * LOAD POINTERS WITH SCREEN PARAMETERS:
257                 *
258                 * Set up vector area 0084-0098 with variables
259                 * describing the current state of the screen
260                 * in the current mode.
261                 *
262 E545 37           VARS   STC
263 E546 F5                 PUSH  PSW
264 E547 C5                 PUSH  B
265 E548 D5                 PUSH  D
266 E549 E5                 PUSH  H
267 E54A E5                 PUSH  H
268 E54B E5                 PUSH  H
269 E54C 2A8000             LHLD  :0080     Get addr 1st byte screen RAM
270 E54F 44                 MOV   B,H       in BC
271 E550 4D                 MOV   C,L
272 E551 E1                 POP   H         Get startaddr table
273 E552 79                 MOV   A,C       )
274 E553 96                 SUB   M         )
275 E554 5F                 MOV   E,A       ) Calc end area used in new
276 E555 23                 INX   H         ) mode. Store it in DE.
277 E556 78                 MOV   A,B       )
278 E557 9E                 SBB   M         )
279 E558 57                 MOV   D,A       )
280 E559 210000             LXI   H,:0000
281 E55C DA60E5             JC    :E560     Jump if insufficient space
282 E55F EB                 XCHG
283 E560 37          VRS05  STC
284 E561 CDA6E5             CALL  :E5A6     Make room for new mode
285 E564 D296E5             JNC   :E596     Jump if no room available
286 E567 2A8800             LHLD  :0088     Get old addr after end
287                                         graphics area
288 E56A 229900             SHLD  :0099     and save it
289 E56D 2A8A00             LHLD  :008A     Get old startaddr char area
290 E570 229B00             SHLD  :009B     and save it
291
292                 * Set up area 0084-0093:
293
294 E573 118400             LXI   D,:0084   Start of variables which
295                                         need offsets
296 E576 2E08               MVI   L,:08     Nr of pointers to be set
297 E578 E3          VRS10  XTHL            Get addr screen parameters
298 E579 79                 MOV   A,C
299 E57A 96                 SUB   M         Calc lobyte
300 E57B 12                 STAX  D         And store it in pointer
301 E57C 23                 INX   H         Next byte
302 E57D 13                 INX   D
303 E57E 78                 MOV   A,B
304 E57F 9E                 SBB   M         Calc hibyte
305 E580 12                 STAX  D         And store it in pointer
306 E581 23                 INX   H
307 E582 13                 INX   D
308 E583 E3                 XTHL
309 E584 2D                 DCR   L         Decr counter
310 E585 C278E5             JNZ   :E578     Next parameter
```

```
312                     * Set up area 0094-0098:
313
314 E588 E1                     POP   H         Get addr 1st parameter
315 E589 0605                   MVI   B,:05     Nr unadjusted constant bytes
316 E58B 7E             VRS20   MOV   A,M       Get parameter
317 E58C 12                     STAX  D         and store it in pointer
318 E58D 23                     INX   H
319 E58E 13                     INX   D
320 E58F 05                     DCR   B         Decr counter
321 E590 C28BE5                 JNZ   :E58B     Next parameter
322 E593 C32EE1                 JMP   :E12E     Popall, CY=0, ret
323
324                     * If no room available:
325
326 E596 E1             VRS30   POP   H
327 E597 C338E1                 JMP   :E138     Popall (CY=1), ret
328                     *
329                     * VECTORS TO TABLES SCREEN PARAMETERS:
330                     *
331                     * The startaddresses of the tables with para-
332                     * meters for the graphic modes are given.
333                     *
334 E59A 45E0            TABM    DBL   :E045     mode 1/2
335 E59C 6FE0                    DBL   :E06F     mode 3/4
336 E59E 99E0                    DBL   :E099     mode 5/6
337                     *
338 E5A0 5AE0            TABMA   DBL   :E05A     mode 1A/2A
339 E5A2 84E0                    DBL   :E084     mode 3A/4A
340 E5A4 AEE0                    DBL   :E0AE     mode 5A/6A
341                     *
342                     ****************************************
343                     * PERFORM MEMORY MANAGEMENT ROUTINE *
344                     ****************************************
345                     *
346                     * Entry: HL points to last free byte in RAM.
347                     *
348 E5A6 23              SMKRM   INX   H
349 E5A7 E5                     PUSH  H
350 E5A8 2AC400                 LHLD  :00C4     Get addr mem.management
351                                             routine
352 E5AB E3                     XTHL            Put it on stack
353 E5AC C9                     RET             Perform this routine and
354                                             return afterwards to origin.
355                                             returnaddress.
356                     *
357                     ************************************
358                     * SET UP AN EMPTY GRAPHICS AREA *
359                     ************************************
360                     *
361                     * Initialises an area of the screen into graphic
362                     * state and blanks it. In 16-colour modes, all
363                     * pixels are set 'on'. The foreground colour is
364                     * the first COLORG colour, the background is black.
365                     *
366                     * Entry: D:  Mode code.
367                     *        C:  Number of graphic lines (1-256).
368                     *        HL: Start of area.
369                     * Exit:  All registers preserved.
370                     *
371 E5AD F5              SGINIT  PUSH  PSW
372 E5AE C5                     PUSH  B
373 E5AF D5                     PUSH  D
```

```
374 E5B0 E5                 PUSH  H
375 E5B1 7A                 MOV   A,D       Mode in A
376 E5B2 E5                 PUSH  H
377 E5B3 110000             LXI   D,:0000   8 blobs 1st graph colour
378 E5B6 2E00               MVI   L,:00     Control byte: graph, low
379                                         def, 4-colour
380 E5B8 1F                 RAR
381 E5B9 1F                 RAR
382 E5BA DACBE5             JC    :E5CB     Jump if 4-colour mode
383
384                 * 16-colour mode only:
385
386 E5BD F5                 PUSH  PSW
387 E5BE 3A9E00             LDA   :009E     Get 1st colour
388 E5C1 87                 ADD   A         )
389 E5C2 87                 ADD   A         ) Move lonibble into
390 E5C3 87                 ADD   A         ) hinibble
391 E5C4 87                 ADD   A         )
392 E5C5 57                 MOV   D,A       Result in D
393 E5C6 1EFF               MVI   E,:FF     8 blobs foreground
394 E5C8 F1                 POP   PSW
395 E5C9 2E80               MVI   L,:80     Control byte: graph, low
396                                         def, 16-colour
397 E5CB 1F         SGI10   RAR
398 E5CC DADEE5             JC    :E5DE     Jump if mode 3/4
399
400                 * Mode 1/2 and 5/6 only:
401
402 E5CF 1F                 RAR
403 E5D0 260B               MVI   H,:0B     Low def fields/line
404 E5D2 3E03               MVI   A,:03     Low def bit mask
405 E5D4 D2E2E5             JNC   :E5E2     Jump if mode 1/2
406
407                 * Mode 5/6 only:
408
409 E5D7 262C               MVI   H,:2C     Super def fields/line
410 E5D9 3E20               MVI   A,:20     Super def bit mask
411 E5DB C3E2E5             JMP   :E5E2
412
413                 * Mode 3/4 only:
414
415 E5DE 2616         SGI20   MVI   H,:16     High def fields/line
416 E5E0 3E11               MVI   A,:11     High def bit mask
417                 *
418 E5E2 B5         SGI30   ORA   L         Add def bits to get mode
419                                         code
420 E5E3 47                 MOV   B,A
421 E5E4 7C                 MOV   A,H       Line length in A
422 E5E5 E1                 POP   H         Get top of area
423 E5E6 F5         SGI50   PUSH  PSW       Save line length
424 E5E7 70                 MOV   M,B       Load line control byte
425 E5E8 2B                 DCX   H
426 E5E9 3640               MVI   M,:40     Null line colour byte
427 E5EB 2B                 DCX   H
428 E5EC 73         SGI60   MOV   M,E       ) Load screen data locations
429 E5ED 2B                 DCX   H         ) with 1 blank field
430 E5EE 72                 MOV   M,D       )
431 E5EF 2B                 DCX   H
432 E5F0 3D                 DCR   A         Next screen location
433 E5F1 C2ECE5             JNZ   :E5EC     Jump if line not ready
434 E5F4 F1                 POP   PSW       Restore nr of locations in A
435 E5F5 0D                 DCR   C         Next screen line
```

```
436 E5F6 C2E6E5              JNZ   :E5E6     Jump if not ready
437 E5F9 C338E1              JMP   :E138     Popall, ret
438                  *
439                  *
440                  *
441 E5FC                     END
```

****
* SYMBOL TABLE *
****

SGI10	E5CB	SGI20	E5DE	SGI30	E5E2	SGI50	E5E6
SGI60	E5EC	SGINIT	E5AD	SMA10	E4F9	SMKRM	E5A6
SS010	E42D	SS015	E438	SS020	E43C	SSM	E45F
SSM0	E407	SSM10	E47F	SSM21	E485	SSMA	E4B6
SSMG	E43E	TABM	E59A	TABMA	E5A0	TABP	E539
VARS	E545	VRS05	E560	VRS10	E578	VRS20	E58B
VRS30	E596						

```
002                             ORG   :E5FC
003                     *
004                     *
005                     *
006                     ********************************
007                     * INITIALISE HEADER / TRAILER *
008                     ********************************
009                     *
010                     * Sets up 4 background colour lines which can act
011                     * as header/trailer. Sets up colours in colour RAM.
012                     * The header/trailer area consists of 4 groups of
013                     * 4 bytes: 00 00 xx 3x, in which xx is the colour
014                     * and 3x the mode word:
015                     *         x=6: Header.
016                     *         x=F: Trailer.
017                     *         x=0: Middle area (split mode).
018                     *
019                     * Entry: HL: 1st byte header/trailer area.
020                     *         DE: Address table with required colours.
021                     *         A : Screen mode.
022                     *         B : Depth -1 in scans of each blanking
023                     *             line: 06 (header), 0F (trailer),
024                     *             00 (middle).
025                     * Exit:   HL: Points after header/trailer area.
026                     *         AFBCDE preserved.
027                     *
028 E5FC F5             SSUBL   PUSH  PSW
029 E5FD C5                     PUSH  B
030 E5FE D5                     PUSH  D
031 E5FF 4F                     MOV   C,A       Store screen mode in C
032 E600 78                     MOV   A,B
033 E601 F630                   ORI   :30       Set 4 colour to make mode
034                                             word + rept.count
035 E603 F5                     PUSH  PSW       Save mode word on stack
036 E604 0600                   MVI   B,:00
037 E606 7B                     MOV   A,E       Get lobyte addr colours
038 E607 D67C                   SUI   :7C
039 E609 CA1EE6                 JZ    :E61E     If char mode then 4 colours
040 E60C 79                     MOV   A,C       Get screen mode
041 E60D 1F                     RAR
042 E60E 1F                     RAR
043 E60F 48                     MOV   C,B
044 E610 DA1FE6                 JC    :E61F     Jump if 4-colour mode
045
046                     * 16-colour modes only:
047
048 E613 F1                     POP   PSW       Restore header/trailer info
049 E614 F680                   ORI   :80       Set 16-colour (msb=1)
050 E616 F5                     PUSH  PSW
051 E617 1A                     LDAX  D         Get colour
052 E618 87                     ADD   A         )
053 E619 87                     ADD   A         ) Move lonibble into
054 E61A 87                     ADD   A         ) hinibble
055 E61B 87                     ADD   A         )
056 E61C 06FF                   MVI   B,:FF     Set all foreground
057 E61E 4F             SBL10   MOV   C,A       Colour in C
058
059                     * Load header/trailer:
060
061 E61F F1             SBL20   POP   PSW       Get mode word
062 E620 F5                     PUSH  PSW
                                MOV   M,A       Load 1st byte pointer
```

```
064 E622 2B                 DCX   H          Next addr in block
065 E623 1A                 LDAX  D          Get colour info
066 E624 13                 INX   D
067 E625 77                 MOV   M,A        Load 2nd byte
068 E626 2B                 DCX   H          Next addr in block
069 E627 70                 MOV   M,B        Load 3rd byte
070 E628 2B                 DCX   H
071 E629 71                 MOV   M,C        Load 4th byte
072 E62A 2B                 DCX   H
073 E62B FEB0               CPI   :B0        All blocks done ?
074 E62D DA1FE6             JC    :E61F      Next one if not
075 E630 F1                 POP   PSW
076 E631 D1                 POP   D
077 E632 C1                 POP   B
078 E633 F1                 POP   PSW
079 E634 C9                 RET
080                 *
081                 ******************************
082                 * SET UP A 4-LINE TEXT AREA *
083                 ******************************
084                 *
085                 * Locates the last few lines of text on the
086                 * screen. If the screen was in split mode, the
087                 * whole contents of the old screen is located.
088                 * If it was mode 0, the last few lines above
089                 * and the cursor line are located.
090                 * The text is then moved to a required position,
091                 * including the cursor, etc.
092                 *
093                 * Entry: HL: Points to address where the text to
094                 *            be put.
095                 * Exit:  HL: Points to new top of text.
096                 *        AFBCDE preserved.
097                 *
098 E635 F5         SMVTXT  PUSH  PSW
099 E636 C5                 PUSH  B
100 E637 D5                 PUSH  D
101 E638 44                 MOV   B,H        ) New top of text in BC
102 E639 4D                 MOV   C,L        )
103 E63A 2A9B00             LHLD  :009B      Get previous start char
104 E63D E5                 PUSH  H          on stack
105 E63E EB                 XCHG             and in DE
106 E63F 21E8FD             LXI   H,:FDE8    Length split screen char
107                                          area
108 E642 19                 DAD   D          Calc 1st line mode byte
109                                          outside screen frame
110 E643 EB                 XCHG             in DE
111 E644 2A7200             LHLD  :0072      Get cursor pos addr
112 E647 CDFBE6             CALL  :E6FB      Check if cursor is still
113                                          inside frame
114 E64A DA75E6             JC    :E675      Jump if not
115 E64D EB                 XCHG             HL is addr 1st line mode
116                                          outside screen frame
117 E64E D1                 POP   D          DE is prev start char
118 E64F E5         SMV10   PUSH  H          Save end preserved text area
119 E650 2A7200             LHLD  :0072      Get cursor pos addr
120 E653 CD6BE3             CALL  :E36B      Delete cursor
121 E656 E3                 XTHL             Previous start char in HL
122 E657 CDC2E6             CALL  :E6C2      Roll screen area to new top
123                                          of text; cursor on last line
124 E65A E3                 XTHL             Cursor pos in HL
125 E65B CDF2E6             CALL  :E6F2      Calc cursor pos against new
```

```
126                                             frame start
127 E65E 09             DAD   B                 Calc new cursor pos addr
128 E65F CD30E3         CALL  :E330             Keep cursor on same pos on
129                                             line
130 E662 2A7800         LHLD  :0078             Get old start line pointer
131 E665 CDF2E6         CALL  :E6F2             HL=HL-DE
132 E668 09             DAD   B                 Calc new cursor pos
133 E669 CD98E6         CALL  :E698             Store addr line mode byte
134                                             current line and last addr
135                                             on that line
136 E66C E1             POP   H                 Get end preserved text area
137 E66D CDF2E6         CALL  :E6F2             HL=HL-DE
138 E670 09             DAD   B
139 E671 D1             POP   D
140 E672 C1             POP   B
141 E673 F1             POP   PSW
142 E674 C9             RET
143
144                 * Scroll frame 1 line if cursor outside frame:
145
146 E675 E1         SMV20   POP   H
147 E676 2A7800             LHLD  :0078         Get startaddr cursor line
148 E679 117AFF             LXI   D,:FF7A
149 E67C 19                 DAD   D             HL: start line after cursor
150 E67D E5                 PUSH  H
151 E67E 111802             LXI   D,:0218
152 E681 19                 DAD   D             Subtract 4 lines and get
153 E682 EB                 XCHG                line mode byte in DE
154 E683 E1                 POP   H             Get end reqd area
155 E684 C34FE6             JMP   :E64F
156                 *
157                 ************************************
158                 * PLACE CURSOR AT BEGIN OF LINE *
159                 ************************************
160                 *
161                 * Sets the cursor at the beginning of a line.
162                 * Several pointers are updated.
163                 *
164                 * SSETC: Given a pointer to the start of line,
165                 *        sets start and end line variables and
166                 *        places the cursor at the beginning of
167                 *        the line.
168                 * SSETL: Sets only start and end line positions.
169                 *
170                 * Entry: HL: Address line mode byte current line.
171                 * Exit:  ABCDEHL preserved.
172                 *
173 E687 F5         SSETC   PUSH  PSW
174 E688 D5                 PUSH  D
175 E689 E5                 PUSH  H
176 E68A 11F8FF             LXI   D,:FFF8
177 E68D 19                 DAD   D             Get addr 1st data byte
178                                             on current line
179 E68E CD30E3             CALL  :E330         Put cursor on screen
180 E691 AF                 XRA   A
181 E692 327B00             STA   :007B         No extended lines
182 E695 E1                 POP   H
183 E696 D1                 POP   D
184 E697 F1                 POP   PSW
185 E698 F5         SSETL   PUSH  PSW
186 E699 227800             SHLD  :0078         Store addr line mode byte
187                                             current line
```

```
188 E69C 3E80              MVI   A,:80     ) Calc lobyte last addr
189 E69E 85                ADD   L         ) on this line
190 E69F 327A00            STA   :007A     Store it in LNEND
191 E6A2 F1                POP   PSW
192 E6A3 C9                RET
193                  *
194                  ************************
195                  * SET GRAPHICS COLOURS *
196                  ************************
197                  *
198                  * Sets the colours available in a 4 colour mode
199                  * and the initial background in a 16 colour mode.
200                  *
201                  * Entry: HL: Points to colours vector.
202                  * Exit:  All registers preserved.
203                  *
204 E6A4 F5           SCOLG  PUSH  PSW
205 E6A5 C5                  PUSH  B
206 E6A6 D5                  PUSH  D
207 E6A7 E5                  PUSH  H
208 E6A8 119E00            LXI   D,:009E   Addr 1st COLORG byte
209 E6AB CD54E2            CALL  :E254     Set COLORG parameters
210 E6AE 3A9D00            LDA   :009D     Get current screen mode
211 E6B1 B7                ORA   A         Check mode type
212 E6B2 2A8200            LHLD  :0082     Get addr after header
213 E6B5 F467E2            CP    :E267     If not mode 0: Load COLORG
214                                          parameters in header
215 E6B8 1F                RAR             Check if char mode
216 E6B9 2A8E00            LHLD  :008E     Get addr after trailer
217 E6BC D467E2            CNC   :E267     If all graphics mode:
218                                          Load colours in trailer
219 E6BF C338E1      SGC10 JMP   :E138     Popall, ret
220                  *
221                  **********************
222                  * MOVE SCREEN AREA *
223                  **********************
224                  *
225                  * Moves a block of screen data from any position
226                  * to any other.
227                  *
228                  * Entry: BC: Points to highaddress target area.
229                  *        DE: Points to highaddress source area.
230                  *        HL: Points to lowaddress -1 source area.
231                  * Exit:  All registers preserved.
232                  *
233 E6C2 F5           MOVES  PUSH  PSW
234 E6C3 C5                  PUSH  B
235 E6C4 D5                  PUSH  D
236 E6C5 E5                  PUSH  H
237 E6C6 CDF2E6            CALL  :E6F2     Calc length of block
238                                          (neg.value)
239 E6C9 7B                MOV   A,E
240 E6CA 91                SUB   C
241 E6CB 7A                MOV   A,D
242 E6CC 98                SBB   B
243 E6CD DAE2E6            JC    :E6E2     Jump if move up
244
245                  * Move down:
246
247 E6D0 54                MOV   D,H       ) Length in DE
248 E6D1 5D                MOV   E,L       )
249 E6D2 09                DAD   B         HL = lowest targetaddr -1
```

```
250 E6D3 C1                 POP   B          BC = lowest sourceaddr -1
251 E6D4 C5                 PUSH  B
252 E6D5 7A          MVS10  MOV   A,D
253 E6D6 B3                 ORA   E
254 E6D7 CAEFE6             JZ    :E6EF      Abort if ready
255 E6DA 13                 INX   D
256 E6DB 23                 INX   H
257 E6DC 03                 INX   B
258 E6DD 0A                 LDAX  B          Get byte from source area
259 E6DE 77                 MOV   M,A        and move it into target area
260 E6DF C3D5E6             JMP   :E6D5      Next one
261
262                  * Move up:
263
264 E6E2 7C          MVS20  MOV   A,H
265 E6E3 B5                 ORA   L
266 E6E4 CAEFE6             JZ    :E6EF      Quit if ready
267 E6E7 23                 INX   H
268 E6E8 1A                 LDAX  D          Get byte from source area
269 E6E9 02                 STAX  B          Move it into target area
270 E6EA 0B                 DCX   B
271 E6EB 1B                 DCX   D
272 E6EC C3E2E6             JMP   :E6E2      Next one
273                  *
274 E6EF C338E1      MVS40  JMP   :E138      Popall, ret
275                  *
276                  ******************
277                  * HL = HL - DE *
278                  ******************
279                  *
280                  * Entry: None.
281                  * Exit:  HL=HL-DE.
282                  *        Other registers preserved.
283                  *
284 E6F2 F5          SUBDE  PUSH  PSW
285 E6F3 7D                 MOV   A,L
286 E6F4 93                 SUB   E
287 E6F5 6F                 MOV   L,A        L=L-E
288 E6F6 7C                 MOV   A,H
289 E6F7 9A                 SBB   D
290 E6F8 67                 MOV   H,A        H=H-D
291 E6F9 F1                 POP   PSW
292 E6FA C9                 RET
293                  *
294                  ********************
295                  * COMPARE HL - DE *
296                  ********************
297                  *
298                  * Compares HL with DE (HL-DE).
299                  *
300                  * Exit: Z=0: Not identical:
301                  *            CY=0: DE < HL.
302                  *            CY=1: DE > HL.
303                  *       Z=1: Identical.
304                  *       AF corrupted, BCDEHL preserved.
305                  *
306 E6FB 7C          COMP   MOV   A,H
307 E6FC 92                 SUB   D
308 E6FD C0                 RNZ
309 E6FE 7D                 MOV   A,L
310 E6FF 93                 SUB   E
311 E700 C9                 RET
```

```
                        *
                        ***************************
                        * ADD OFFSET TO ADDRESS *
                        ***************************
                        *
                        * Sets HL = HL + A.
                        *
                        * Entry: HL: baseaddress.
                        *        A : offset.
                        * Exit:  HL = HL + A.
                        *        BCDE preserved.
                        *
E701 85                 DADA    ADD   L          Add lobyte addr to offset
E702 6F                         MOV   L,A        and store it in L
E703 D0                         RNC
E704 24                         INR   H          Incr hibyte if overflow
E705 C9                         RET
                        *
                        ************************************
                        * TWO COMPLEMENT OF 16-BITS DATA *
                        ************************************
                        *
                        * Sets HL = - HL.
                        *
                        * Entry: Data to be complemented in HL.
                        * Exit:  HL contains two-complement.
                        *        AFBCDE preserved.
                        *
E706 F5                 CMPHL   PUSH  PSW
E707 7D                         MOV   A,L
E708 2F                         CMA              Compl. L
E709 6F                         MOV   L,A        and store it
E70A 7C                         MOV   A,H
E70B 2F                         CMA              Compl. H
E70C 67                         MOV   H,A        and store it
E70D 23                         INX   H          Add 1
E70E F1                         POP   PSW
E70F C9                         RET
                        *
                        ******************************
                        * DRAW A DOT ON THE SCREEN *
                        ******************************
                        *
                        * Draws a single blob of a colour anywhere
                        * on the screen.
                        *
                        * Entry: C,HL: Y,X coordinate of the dot.
                        *        A:    Colour of the dot.
                        * Exit:  CY=0: O.K.
                        *        CY=1: Error code in A.
                        *        ABCDEHL preserved.
                        *
E710 B7                 SDOT    ORA   A
E711 F5                         PUSH  PSW
E712 C5                         PUSH  B
E713 D5                         PUSH  D
E714 E5                         PUSH  H
E715 41                         MOV   B,C        Y-coord in B
E716 54                         MOV   D,H        ) X-coord in DE
E717 5D                         MOV   E,L        )
E718 C31DE8                     JMP   :E81D      Into 'SFILL'
                        *
```

```
                        ******************************
                        * DRAW A LINE ON THE SCREEN *
                        ******************************
                        *
                        * Draws a line in a given colour between two
                        * arbitrary points on the screen.
                        *
                        * The coordinates are given inclusively. The
                        * line will be drawn starting at the left end,
                        * whichever order the parameters are given in.
                        *
                        * Entry: B,DE: Y,X coordinate of one end of the
                        *              line.
                        *        C,HL: Idem of the other end.
                        *        A:    Colour of the line.
                        * Exit:  CY=0: O.K.
                        *        CY=1: Errorcode in A.
                        *        ABCDEHL preserved.
                        *
E71B B7                 SDRAW   ORA   A
E71C F5                         PUSH  PSW
E71D C5                         PUSH  B
E71E D5                         PUSH  D
E71F E5                         PUSH  H
E720 CD3AE8                     CALL  :E83A     Check arguments, set colour
E723 F5                         PUSH  PSW
E724 CDFBE6                     CALL  :E6FB     Check direction of line
E727 3E00                       MVI   A,:00     Set 'no X,Y swop'
E729 D22EE7                     JNC   :E72E     Jump if X > Y

                        * Swop X,Y:

E72C EB                         XCHG            Exchange X coordinates
E72D 2F                         CMA
                        *
E72E 32C000             DRL30   STA   :00C0     FF if X,Y swop, else 00
E731 79                         MOV   A,C
E732 E607                       ANI   :07
E734 57                         MOV   D,A       Offset in field in D
E735 F1                         POP   PSW       Get Y-pos left end
E736 E5                         PUSH  H         Save X length
E737 CDB9EB                     CALL  :EBB9     Pntr to start of line in
                                                screen RAM
E73A E3                         XTHL
E73B D5                         PUSH  D         Save offset
E73C E5                         PUSH  H         Save DX
E73D CD06E7                     CALL  :E706     HL = - DX
E740 E3                         XTHL
E741 E5                         PUSH  H         Save DX
E742 6B                         MOV   L,E
E743 2600                       MVI   H,:00
E745 29                         DAD   H
E746 22B900                     SHLD  :00B9     Store 2*DY (adj long
                                                sectors)
E749 7B                         MOV   A,E       DY in A
E74A 32BD00                     STA   :00BD     Set count of sectors (1-256)
E74D E1                         POP   H
E74E CD60EB                     CALL  :EB60     HL = DX / DY
E751 22BB00                     SHLD  :00BB     SECT is INT(DX/DY)
E754 C1                         POP   B         Get -DX
E755 E5                         PUSH  H         Save SECT
E756 7B                         MOV   A,E
```

```
436 E757 CD46EB             CALL  :EB46     HL = SECT*DY
437 E75A 09                 DAD   B         HL = SECT*DY-DX
438 E75B 29                 DAD   H         HL = 2*(SECT*DY-DX)
439 E75C 22B500             SHLD  :00B5     Store amount to add into
440                                         count
441 E75F E1                 POP   H         Get SECT
442 E760 7C                 MOV   A,H
443 E761 1F                 RAR
444 E762 7D                 MOV   A,L
445 E763 1F                 RAR             A=INT(SECT/2)
446 E764 32BE00             STA   :00BE     Store amount to trim off
447                                         last sector
448 E767 3C                 INR   A
449 E768 6F                 MOV   L,A
450 E769 2600               MVI   H,:00     HL = INIT
451 E76B E5                 PUSH  H
452 E76C 29                 DAD   H         HL = 2*INIT
453 E76D 7B                 MOV   A,E
454 E76E CD46EB             CALL  :EB46     HL = 2*INIT*DY
455 E771 09                 DAD   B         HL = 2*INIT*DY-DX
456 E772 22B700             SHLD  :00B7     Set INIT running total
457 E775 E1                 POP   H         Get length 1st sector
458 E776 F1                 POP   PSW
459 E777 4F                 MOV   C,A       C is initial offset
460 E778 7B                 MOV   A,E
461 E779 B7                 ORA   A
462 E77A C284E7             JNZ   :E784     If more than 1 sector
463 E77D 32BE00             STA   :00BE     Store amount to trim off
464                                         last sector
465 E780 2ABB00             LHLD  :00BB     Get lower of 2 possible
466                                         sectors
467 E783 23                 INX   H         Frig length if only 1 sector
468 E784 11BD00     DRL40   LXI   D,:00BD   Addr of nr of sectors
469 E787 1A                 LDAX  D         Get count of sectors
470 E788 D601               SUI   :01       -1
471 E78A 12                 STAX  D         Store it again
472 E78B D297E7             JNC   :E797     Jump if not last sector
473
474                 * Trim off last sector:
475
476 E78E 3ABE00             LDA   :00BE     Get amount to trim off
477                                         last sector
478 E791 2F                 CMA
479 E792 5F                 MOV   E,A
480 E793 16FF               MVI   D,:FF
481 E795 13                 INX   D
482 E796 19                 DAD   D
483                 *
484 E797 110100     DRL50   LXI   D,:0001   Init Y-size is 1
485 E79A 3AC000             LDA   :00C0     ) Check if swop X,Y dir.
486 E79D B7                 ORA   A         ) (line > 45 degrees)
487 E79E CAA2E7             JZ    :E7A2     Jump if not
488 E7A1 EB                 XCHG            Swop X,Y direction
489 E7A2 43         DRL60   MOV   B,E       Get Y-size in B
490 E7A3 EB                 XCHG            X-size in DE
491 E7A4 E1                 POP   H         Get memory pointer
492 E7A5 05                 DCR   B
493 E7A6 3ABF00             LDA   :00BF     ) Check for Y-invert
494 E7A9 B7                 ORA   A         )
495 E7AA F5                 PUSH  PSW       Save condition
496 E7AB C4FCE7             CNZ   :E7FC     Move pntr to bottom of
497                                         sector
```

```
498 E7AE 1B                     DCX   D         Interfacing
499 E7AF CDF7EA                 CALL  :EAF7     Draw next sector of line
500 E7B2 13                     INX   D         ) Re-instate real values
501 E7B3 04                     INR   B         )
502 E7B4 F1                     POP   PSW       Get earlier condition
503 E7B5 CABAE7                 JZ    :E7BA     Jump if no Y-invert
504 E7B8 0601                   MVI   B,:01     Init 1 blob down only
505 E7BA CDFCE7       DRL80     CALL  :E7FC     Move ptr up/down
506 E7BD 79                     MOV   A,C       Get offset
507 E7BE 83                     ADD   E         Add X-movement
508 E7BF 4F                     MOV   C,A       ) Save result
509 E7C0 47                     MOV   B,A       )
510 E7C1 E607                   ANI   :07
511 E7C3 B9                     CMP   C
512 E7C4 CAD2E7                 JZ    :E7D2     Jump if not new field
513
514                   * If new field:
515
516 E7C7 4F                     MOV   C,A       Update offset
517 E7C8 78                     MOV   A,B       Get complement new offset
518 E7C9 A9                     XRA   C         Clip bits 0,1,2 off
519 E7CA 1F                     RAR             )
520 E7CB 1F                     RAR             )
521 E7CC 2F                     CMA             ) Update pointer
522 E7CD 5F                     MOV   E,A       ) to new field
523 E7CE 16FF                   MVI   D,:FF     )
524 E7D0 13                     INX   D         )
525 E7D1 19                     DAD   D         )
526                   *
527 E7D2 E5           DRL83     PUSH  H         Save memory pointer
528 E7D3 2AB500                 LHLD  :00B5     Get amount to add into count
529 E7D6 EB                     XCHG            in DE
530 E7D7 2AB700                 LHLD  :00B7     Get count running total
531 E7DA 19                     DAD   D         Add up
532 E7DB EB                     XCHG            Result in DE
533 E7DC 2ABB00                 LHLD  :00BB     Get lowest of 2 possible
534                                             sectors
535 E7DF EB                     XCHG            in DE (distance to go)
536 E7E0 7C                     MOV   A,H
537 E7E1 B7                     ORA   A
538 E7E2 F2EDE7                 JP    :E7ED     Jump if short sector
539
540                   * If long sector:
541
542 E7E5 13                     INX   D         Go one blob further
543 E7E6 D5                     PUSH  D
544 E7E7 EB                     XCHG
545 E7E8 2AB900                 LHLD  :00B9     Get adjustment for long
546                                             sectors
547 E7EB 19                     DAD   D         Adjust error term
548 E7EC D1                     POP   D
549 E7ED 22B700       DRL86     SHLD  :00B7     Update running total
550 E7F0 EB                     XCHG
551 E7F1 3ABD00                 LDA   :00BD     Get nr of sectors
552 E7F4 3C                     INR   A
553 E7F5 C284E7                 JNZ   :E784     Next sector if not ready
554
555                   * If ready:
556
557 E7F8 E1                     POP   H
558 E7F9 C338E1                 JMP   :E138     Popall, ret
```

```
560                     *
561                     *
562 E7FC                        END
```

```
****************************
* S Y M B O L   T A B L E *
****************************
```

CMPHL	E706	COMP	E6FB	DADA	E701	DRL30	E72E
DRL40	E784	DRL50	E797	DRL60	E7A2	DRL80	E7BA
DRL83	E7D2	DRL86	E7ED	MOVES	E6C2	MVS10	E6D5
MVS20	E6E2	MVS40	E6EF	SBL10	E61E	SBL20	E61F
SCOLG	E6A4	SDOT	E710	SDRAW	E71B	SGC10	E6BF
SMV10	E64F	SMV20	E675	SMVTXT	E635	SSETC	E687
SSETL	E698	SSUBL	E5FC	SUBDE	E6F2		

```
002                             ORG   :E7FC
003                     *
004                     *
005                     *
006                     ****************************
007                     * MOVE POINTER UP / DOWN *
008                     ****************************
009                     *
010                     * Subroutine of SDRAW (2E71B).
011                     * Takes a pointer to screen and moves it up or
012                     * down the screen a number of lines. The move
013                     * direction depends on DIRN1 (00BF).
014                     *
015                     * Entry: HL: Pointer.
016                     *        B:  Number of lines.
017                     * Exit:  HL: updated.
018                     *        AFBCDE preserved.
019                     *
020 E7FC F5             UPDTP   PUSH  PSW
021 E7FD D5                     PUSH  D
022 E7FE E5                     PUSH  H
023 E7FF 3A9800                 LDA   :0098     Get number bytes/line
024 E802 6F                     MOV   L,A       ) Store it in HL
025 E803 2600                   MVI   H,:00     )
026 E805 78                     MOV   A,B       Get nr of lines
027 E806 CD46EB                 CALL  :EB46     Calc total length in HL
028 E809 3ABF00                 LDA   :00BF     Get Y-direction
029 E80C B7                     ORA   A         Test if up or down
030 E80D C406E7                 CNZ   :E706     If down:  Calc 2-compl of HL
031 E810 D1                     POP   D
032 E811 19                     DAD   D         Update pntr
033 E812 CDC7EA                 CALL  :EAC7     Into or out archive area
034 E815 D1                     POP   D
035 E816 F1                     POP   PSW
036 E817 C9                     RET
037                     *
038                     ******************************************
039                     * FILL A RECTANGULAR AREA ON THE SCREEN *
040                     ******************************************
041                     *
042                     * Fills an arbitrary rectangle with a given colour.
043                     *
044                     * The middle of the rectangle is filled first, then
045                     * the left and then the right edge vertical strips.
046                     * The coordinates are given inclusively.
047                     * The rectangle is filled in the same order, which
048                     * ever order the parameters are given in.
049                     *
050                     * Entry: B,DE: Y,X coordinate of one corner.
051                     *        C,HL: Idem of the opposite corner.
052                     *        A:    Colour.
053                     * Exit:  ABCDEHL preserved.
054                     *        CY=0: OK.
055                     *        CY=1: A contains error code.
056                     *
057 E818 B7             SFILL   ORA   A
058 E819 F5                     PUSH  PSW
059 E81A C5                     PUSH  B
060 E81B D5                     PUSH  D
061 E81C E5                     PUSH  H
062 E81D CD3AE8         FIL10   CALL  :E83A     Check arguments, get colour
063 E820 57                     MOV   D,A       Get Y-coord left corner
```

```
064 E821 3ABF00          LDA   :00BF     ) Check if Y invert
065 E824 B7              ORA   A         )
066 E825 7A              MOV   A,D
067 E826 CA2AE8          JZ    :E82A     Jump if no Y-inversion
068 E829 93              SUB   E         Y-pos bottom left
069 E82A E5        FIL20 PUSH  H         Save X-size
070 E82B CDB9EB          CALL  :EBB9     Get memory address
071 E82E 79              MOV   A,C
072 E82F E607            ANI   :07
073 E831 4F              MOV   C,A       Offset in C
074 E832 43              MOV   B,E       Height in B
075 E833 D1              POP   D         Width in DE
076 E834 CDF7EA          CALL  :EAF7     Fill block
077 E837 C338E1          JMP   :E138     Popall, ret
078                *
079                ********************************
080                * CHECK ARGUMENTS, GET COLOUR *
081                ********************************
082                *
083                * Checks the arguments given to a entry point and
084                * sets up the colour variables. Swops order of two
085                * points given if necessary.
086                *
087                * Entry: ABCDEHL: See SFILL.
088                *        All registers and a returnaddr on stack.
089                * Exit:  CY=0: O.K.:
090                *              A,BC:  Set to left corner.
091                *              DE,HL: Set to Y,X lengths of line.
092                *              DIRN1: <>0 if Y-direction negative.
093                *        CY=1: Error report:
094                *              A=1:   Off screen.
095                *              A=2:   Colour not available.
096                *
097 E83A CDC3E9     ARGCHK CALL  :E9C3     Set up colour variables
098 E83D DA75E8            JC    :E875     Jump if colour not av.
099 E840 CD7AEB          CALL  :EB7A     Check if room available
100 E843 DA7FE8          JC    :E87F     Jump if not
101 E846 C5              PUSH  B
102 E847 48              MOV   C,B       Y-coord one point in C
103 E848 EB              XCHG
104 E849 CD7AEB          CALL  :EB7A     Check if room available
105 E84C EB              XCHG
106 E84D C1              POP   B
107 E84E DA7FE8          JC    :E87F     Jump if not
108 E851 CDFBE6          CALL  :E6FB     Compare HL-DE
109 E854 D25DE8          JNC   :E85D     Right most point to C,HL
110
111                * Swop Y-coord.:
112
113 E857 EB              XCHG
114 E858 F5              PUSH  PSW
115 E859 78              MOV   A,B       )
116 E85A 41              MOV   B,C       ) Swop 2 points
117 E85B 4F              MOV   C,A       )
118 E85C F1              POP   PSW
119                *
120 E85D CDF2E6     ARC10  CALL  :E6F2     Calc horizontal length
121 E860 D5              PUSH  D         Save X-pos left corner
122 E861 79              MOV   A,C
123 E862 90              SUB   B
124 E863 1600            MVI   D,:00     Clear Y-invert flag
125 E865 D26BE8          JNC   :E86B     Jump if end above start
```

```
126 E868 2F                     CMA
127 E869 3C                     INR   A
128 E86A 15                     DCR   D
129                     *
130 E86B 5F             ARC20   MOV   E,A       Get vertical length
131 E86C 7A                     MOV   A,D
132 E86D 32BF00                 STA   :00BF     Set Y-invert flag
133 E870 1600                   MVI   D,:00
134 E872 78                     MOV   A,B       Get Y-pos left corner
135 E873 C1                     POP   B         Get X-pos left corner
136 E874 C9                     RET
137
138                     * If colour error:
139
140 E875 3E02           ARC90   MVI   A,:02     Error colour not available
141 E877 E1             ARC98   POP   H
142 E878 E1                     POP   H
143 E879 D1                     POP   D
144 E87A C1                     POP   B
145 E87B 33                     INX   SP
146 E87C 33                     INX   SP
147 E87D 37                     STC             Return error
148 E87E C9                     RET
149
150                     * If off screen error:
151
152 E87F 3E01           ARC99   MVI   A,:01     Error off screen
153 E881 C377E8                 JMP   :E877     Abort
154                     *
155                     ***********************************************
156                     * ASK COLOUR OF A POINT ON THE SCREEN AND *
157                     *     ASK SIZE OF THE GRAPHICS SCREEN     *
158                     ***********************************************
159                     *
160                     * Aks the colour of a given point on the screen
161                     * and the size of the graphics area of the screen.
162                     *
163                     * Entry: C,HL: Y,X coordinate of the dot required.
164                     * Exit:  CY=0: O.K.:
165                     *                A:    Colour at requested point.
166                     *                B,DE: Max. coordinates of the
167                     *                      graphics area.
168                     *                CHL preserved.
169                     *         CY=1: A:    Error code.
170                     *                BCDEHL preserved.
171                     *
172 E884 E5             SSCRN   PUSH  H
173 E885 C5                     PUSH  B
174 E886 CD7AEB                 CALL  :EB7A     Check if room available
175 E889 DAD8E8                 JC    :E8D8     Jump if not
176 E88C 7D                     MOV   A,L
177 E88D E607                   ANI   :07
178 E88F F5                     PUSH  PSW       Remember field offset
179 E890 79                     MOV   A,C       )
180 E891 44                     MOV   B,H       ) Coord in A,B,C
181 E892 4D                     MOV   C,L       )
182 E893 CDB9EB                 CALL  :EBB9     Get address of point
183 E896 3A9D00                 LDA   :009D     Get current screen mode
184 E899 1F                     RAR
185 E89A 1F                     RAR
186 E89B DAB8E8                 JC    :E8B8     Jump if 4-colour mode
```

```
                        * If 16-colour mode:
                        *
E89E CDF6E8                     CALL  :E8F6     Colours to buffer
E8A1 21A300                     LXI   H,:00A3   Addr SCXBUF
E8A4 F1                         POP   PSW
E8A5 CD01E7                     CALL  :E701     Calc addr in buffer
E8A8 7E                         MOV   A,M       Get colour for reqd blob
E8A9 2A9600             SSC10   LHLD  :0096     Get nr of graphics lines
E8AC 45                         MOV   B,L
E8AD 05                         DCR   B         lobyte -1 in B
E8AE 2A9400                     LHLD  :0094     Get nr of hor. blobs
E8B1 2B                         DCX   H         -1
E8B2 EB                         XCHG            in DE
E8B3 E1                         POP   H
E8B4 4D                         MOV   C,L       Y-coord in C
E8B5 E1                         POP   H         X-coord in HL
E8B6 B7                         ORA   A         CY=0
E8B7 C9                         RET
                        *
                        * If 4-colour mode:
                        *
E8B8 56                 SSC30   MOV   D,M       )
E8B9 2B                         DCX   H         ) Get screen data of
E8BA 5E                         MOV   E,M       ) point in DE
E8BB F1                         POP   PSW
E8BC 4F                         MOV   C,A       Field offset in C
E8BD 0601                       MVI   B,:01
E8BF CDE1EB                     CALL  :EBE1     Set mask for bits
E8C2 219E00                     LXI   H,:009E   Pntr to COLORG colours
E8C5 78                         MOV   A,B
E8C6 A2                         ANA   D         Test top bit result
E8C7 CACCE8                     JZ    :E8CC     Skip if 0
E8CA 23                         INX   H
E8CB 23                         INX   H
E8CC 78                 SSC40   MOV   A,B
E8CD A3                         ANA   E         Test bottom bit result
E8CE CAD2E8                     JZ    :E8D2     Skip if 0
E8D1 23                         INX   H
E8D2 7E                 SSC50   MOV   A,M       Get result from table
E8D3 E60F                       ANI   :0F       Colour bits only
E8D5 C3A9E8                     JMP   :E8A9
                        *
                        * If off screen:
                        *
E8D8 E1                 SSC99   POP   H
E8D9 C1                         POP   B
E8DA 3E01                       MVI   A,:01     Error 'off screen'
E8DC 37                         STC
E8DD C9                         RET
                        *
                        ********************
                        * UPDATE A FIELD *
                        ********************
                        *
                        * Given a mask of bits to be changed, a colour to
                        * set them to and a memory address where the field
                        * starts. This routine reads, updates and replaces
                        * a field.
                        *
                        * Entry: HL: Memory address of start of field.
                        *         B: Mask of bits to be changed.
                        *         C: Colour to change to (hinibble).
```

```
250                     * Exit:  All registers preserved.
251                     *
252 E8DE F5             SUPDTE  PUSH  PSW
253 E8DF C5                     PUSH  B
254 E8E0 D5                     PUSH  D
255 E8E1 E5                     PUSH  H
256 E8E2 79                     MOV   A,C       )
257 E8E3 0F                     RRC             )
258 E8E4 0F                     RRC             ) Move hinibble C into
259 E8E5 0F                     RRC             ) lonibble
260 E8E6 0F                     RRC             )
261 E8E7 4F                     MOV   C,A       )
262 E8E8 C5                     PUSH  B
263 E8E9 CDF6E8                 CALL  :E8F6     Get current state of screen
264 E8EC C1                     POP   B
265 E8ED CDB2E9                 CALL  :E9B2     Change as required
266 E8F0 E1                     POP   H
267 E8F1 E5                     PUSH  H
268 E8F2 C3BAC6                 JMP   :C6BA     Set up screen bits for
269                                             mode 1
270                     *
271 E8F5 FF                     DATA  :FF
272                     *
273                     ************************************
274                     * LOAD BUFFER SCXBUF FROM SCREEN *
275                     ************************************
276                     *
277                     * Takes 2 bytes of screen info in 16-colour mode
278                     * and places them in SCXBUF (00A3-AB) in 'standard
279                     * form'.
280                     *
281                     * Entry: HL: Points to 1st byte of info on screen.
282                     * Exit:  All registers corrupted.
283                     *
284 E8F6 23             SSFM    INX   H
285 E8F7 7E                     MOV   A,M       Get previous colour byte
286 E8F8 E60F                   ANI   :0F       Background only
287 E8FA 4F                     MOV   C,A       Previous background in C
288 E8FB 2B                     DCX   H
289 E8FC 56                     MOV   D,M       Select byte in D
290 E8FD 2B                     DCX   H
291 E8FE 5E                     MOV   E,M       Colour byte in E
292 E8FF 2B                     DCX   H
293 E900 E5                     PUSH  H         Save pntr to next field
294                                             select byte
295 E901 7B                     MOV   A,E       Colour byte in A
296 E902 E6F0                   ANI   :F0       )
297 E904 0F                     RRC             ) Foreground colour
298 E905 0F                     RRC             ) in lonibble
299 E906 0F                     RRC             )
300 E907 0F                     RRC             )
301 E908 47                     MOV   B,A       Foreground colour in B
302 E909 21A300                 LXI   H,:00A3   Addr SCXBUF
303 E90C 7A                     MOV   A,D       Get bit mask
304 E90D 1608                   MVI   D,:08     8 bytes to set
305 E90F 07             SSF10   RLC
306 E910 71                     MOV   M,C       Set background
307 E911 D21EE9                 JNC   :E91E     Jump if background
308 E914 70                     MOV   M,B       Else: set foreground
309 E915 F5                     PUSH  PSW
310 E916 7B                     MOV   A,E       Get colour byte
311 E917 E60F                   ANI   :0F       Background colour only
```

```
312 E919 4F              MOV   C,A       Background is current BG
313 E91A 32AC00          STA   :00AC     Store it as colour carried
314                                      out to next field
315 E91D F1              POP   PSW
316 E91E 23        SSF20 INX   H         Next pos in SCXBUF
317 E91F 15              DCR   D         Count -1
318 E920 C20FE9          JNZ   :E90F     Loop if more bits
319 E923 C1              POP   B         Get pntr to next field
320                                      select byte
321 E924 36FF            MVI   M,:FF     Flag no carry out in SBGOU
322 E926 0A              LDAX  B         Get next select byte
323 E927 34        SSF30 INR   M         Set 'carry out' flag
324 E928 07              RLC
325 E929 D227E9          JNC   :E927     Loop counting carry out
326 E92C C9              RET
327                *
328                **********************************
329                * SET UP SCREEN BITS FOR MODE 1 *
330                **********************************
331                *
332                * Takes the 8 blobs represented in standard form
333                * in SCXBUF, and tries to represent them in a way
334                * which the screen requires for mode 1.
335                * Up to 2 colours is easy. 3 require to attempt
336                * to carry in the 1st colour from the previous
337                * byte.
338                *
339                * Entry: HL: Points to 1st of the 2 screen bytes.
340                * Exit:  Screen will be updated as well as
341                *        possible.
342                *        All registers preserved.
343                *
344                SBF00
345 E92D 23        SBFM  INX   H
346 E92E 7E               MOV   A,M
347 E92F F680            ORI   :80
348 E931 5F              MOV   E,A       Prev background in E
349 E932 1600            MVI   D,:00     Init bit map
350 E934 21A300          LXI   H,:00A3   Addr SCXBUF
351 E937 00              NOP
352 E938 00              NOP
353 E939 3AAB00    SBF05 LDA   :00AB     Get flag for colour carried
354                                      out
355 E93C B7              ORA   A
356 E93D CA45E9          JZ    :E945     Jump if no carry out
357 E940 3AAC00          LDA   :00AC     Get colour carried out
358 E943 F680            ORI   :80       Set msb=1
359 E945 4F        SBF10 MOV   C,A       Background colour in C
360 E946 7E              MOV   A,M
361 E947 23              INX   H
362 E948 47              MOV   B,A       Set 1st blob as FG colour
363 E949 7A              MOV   A,D       )
364 E94A 37              STC             ) 1 bit in right end bit
365 E94B 17              RAL             ) mask
366 E94C 57              MOV   D,A       )
367 E94D 7E        SBF30 MOV   A,M
368 E94E 23              INX   H
369 E94F B8              CMP   B         Is next blob FG colour ?
370 E950 37              STC
371 E951 CA61E9          JZ    :E961     Jump if true
372 E954 F680            ORI   :80
373 E956 B9              CMP   C         Test if same as BG ?
```

```
374 E957 CA60E9            JZ    :E960      Jump if true
375 E95A 0D                DCR   C
376 E95B 0C                INR   C
377 E95C FA86E9            JM    :E986      No luck if BG used already
378 E95F 4F                MOV   C,A        Else set it to BG
379 E960 B7        SBF40   ORA   A
380 E961 7A        SBF45   MOV   A,D        )
381 E962 17                RAL              ) New bit in bottom of mask
382 E963 57                MOV   D,A        )
383 E964 7D        SBF50   MOV   A,L
384 E965 FEAB              CPI   :AB        End of buffer reached ?
385 E967 C24DE9            JNZ   :E94D      Loop until all set up
386 E96A E1                POP   H
387 E96B E5                PUSH  H
388 E96C 23                INX   H
389 E96D 7B                MOV   A,E
390 E96E E60F              ANI   :0F
391 E970 5F                MOV   E,A        Only lonibble of E
392 E971 7E                MOV   A,M
393 E972 E6F0              ANI   :F0        Only hinibble of M
394 E974 B3                ORA   E
395 E975 77                MOV   M,A        Add both nibbles together
396 E976 2B                DCX   H
397 E977 78                MOV   A,B
398 E978 87                ADD   A
399 E979 87                ADD   A
400 E97A 87                ADD   A
401 E97B 87                ADD   A          FG colour to top bits
402 E97C 47                MOV   B,A
403 E97D 79                MOV   A,C
404 E97E E60F              ANI   :0F        Low nibble only
405 E980 B0                ORA   B
406 E981 72                MOV   M,D        Bit map from D
407 E982 2B                DCX   H
408 E983 77                MOV   M,A        Colours from E
409 E984 E1        SBF90   POP   H
410 E985 C9                RET
411
412                * 3 colours needed:
413
414 E986 7B        SBF80   MOV   A,E
415 E987 B7                ORA   A
416 E988 F284E9            JP    :E984      Jump if tried BG carried in
417 E98B E60F              ANI   :0F        Previous BG
418 E98D B8                CMP   B          Test against 1st blob colour
419 E98E E1                POP   H
420 E98F E5                PUSH  H
421 E990 23                INX   H
422 E991 23                INX   H
423 E992 7E                MOV   A,M        Get bit map
424 E993 37        SBF83   STC
425 E994 17                RAL
426 E995 D293E9            JNC   :E993      Ignore leading BG
427 E998 CAA1E9            JZ    :E9A1      Jump if colour matches
428                                           anyway
429 E99B 3C                INR   A
430 E99C C284E9            JNZ   :E984      No good if BG used
431
432                * Background not in use:
433
434 E99F 58                 MOV   E,B       Set previous BG
435 E9A0 00                 NOP
```

```
436 E9A1 21A300     SBF85   LXI   H,:00A3   Addr SCXBUF
437 E9A4 1600               MVI   D,:00     Init D
438 E9A6 00                 NOP
439 E9A7 4A                 MOV   C,D       and C (BG free)
440 E9A8 7E         SBF88   MOV   A,M       Get byte from SCXBUF
441 E9A9 B8                 CMP   B
442 E9AA C239E9             JNZ   :E939     Jump if blob not old BG
443                                         colour
444 E9AD 00                 NOP
445 E9AE 23                 INX   H
446 E9AF C3A8E9             JMP   :E9A8     Next blob
447                 *
448                 ************************
449                 * UPDATE BUFFER SCXBUF *
450                 ************************
451                 *
452                 * Takes a set of 'update instructions' in BC and
453                 * sets various bytes in SCXBUF accordingly.
454                 *
455                 * Entry: BC: Instructions.
456                 * Exit:  All registers corrupted.
457                 *
458 E9B2 21A300     SUDCH   LXI   H,:00A3   Addr SCXBUF
459 E9B5 78                 MOV   A,B       Mask in A
460 E9B6 0608               MVI   B,:08     8 byte to be done
461 E9B8 07         SUD10   RLC             Bit from mask into CY
462 E9B9 D2BDE9             JNC   :E9BD     Jump if bit = 0
463 E9BC 71                 MOV   M,C       Else: C into SCXBUF
464 E9BD 23         SUD20   INX   H
465 E9BE 05                 DCR   B
466 E9BF C2B8E9             JNZ   :E9B8     Next byte if not ready
467 E9C2 C9                 RET
468                 *
469                 ****************************
470                 * SET UP COLOUR VARIABLES *
471                 ****************************
472                 *
473                 * Entry: A: Colour.
474                 * Exit:  CY=1: Colour not available.
475                 *        CY=0: O.K.; ABCDEHL preserved.
476                 *
477 E9C3 B7         COLSU   ORA   A
478 E9C4 F5                 PUSH  PSW
479 E9C5 C5                 PUSH  B
480 E9C6 4F                 MOV   C,A       Colour in C
481 E9C7 AF                 XRA   A
482 E9C8 32C100             STA   :00C1     Reset animate flag
483 E9CB 3A9D00             LDA   :009D     Get current screen mode
484 E9CE 1F                 RAR
485 E9CF 1F                 RAR
486 E9D0 D2FDE9             JNC   :E9FD     Jump if 16-colour
487
488                 * If 4-colour:
489
490 E9D3 79                 MOV   A,C       Get colour
491 E9D4 FE10               CPI   :10
492 E9D6 DAE1E9             JC    :E9E1     Jump if < 16
493 E9D9 CD86D8             CALL  :D886     Check if >= 20. Set 00C1
494                                         for 4-colour animate if not
495 E9DC E603       CSU04   ANI   :03
496 E9DE C3E7E9             JMP   :E9E7     Bottom 3 bits only
497 E9E1 CD9BEB     CSU05   CALL  :EB9B     Find colour in COLORG reg
```

```
498                                                    (2 bit code)
499 E9E4 D207EA             JNC   :EA07       Jump if colour not av.
500 E9E7 0600       CSU08   MVI   B,:00
501 E9E9 FE02               CPI   :02
502 E9EB DAEFE9             JC    :E9EF       Jump if top bit 0
503 E9EE 05                 DCR   B           Set 00/FF on top bit
504 E9EF E601       CSU10   ANI   :01
505 E9F1 2F                 CMA
506 E9F2 3C                 INR   A           Set 00/FF on bottom bit
507 E9F3 32C200     CSU30   STA   :00C2       )
508 E9F6 78                 MOV   A,B         ) Store details for colour
509 E9F7 32C300             STA   :00C3       ) reqd
510 E9FA C1                 POP   B
511 E9FB F1                 POP   PSW
512 E9FC C9                 RET
513
514                 * If 16-colour:
515
516 E9FD 79         CSU40   MOV   A,C         Get colour
517 E9FE 87                 ADD   A           )
518 E9FF 87                 ADD   A           ) SHL 8
519 EA00 87                 ADD   A           )
520 EA01 87                 ADD   A           )
521 EA02 06FF               MVI   B,:FF
522 EA04 C3F3E9             JMP   :E9F3       Store details for colour
523                                           reqd
524
525                 * If colour not found:
526
527 EA07 C1         CSU99   POP   B
528 EA08 F1                 POP   PSW
529 EA09 3F                 CMC               Quit; 'colour not available'
530 EA0A C9                 RET
531                 *
532                 *
533                 *
534 EA0B                    END
```

```
****************************
* S Y M B O L   T A B L E *
****************************
```

ARC10	E85D	ARC20	E86B	ARC90	E875	ARC98	E877
ARC99	E87F	ARGCHK	E83A	COLSU	E9C3	CSU04	E9DC
CSU05	E9E1	CSU08	E9E7	CSU10	E9EF	CSU30	E9F3
CSU40	E9FD	CSU99	EA07	FIL10	E81D	FIL20	E82A
SBF00	E92D	SBF05	E939	SBF10	E945	SBF30	E94D
SBF40	E960	SBF45	E961	SBF50	E964	SBF80	E986
SBF83	E993	SBF85	E9A1	SBF88	E9A8	SBF90	E984
SBFM	E92D	SFILL	E818	SSC10	E8A9	SSC30	E8B8
SSC40	E8CC	SSC50	E8D2	SSC99	E8D8	SSCRN	E884
SSF10	E90F	SSF20	E91E	SSF30	E927	SSFM	E8F6
SUD10	E9B8	SUD20	E9BD	SUDCH	E9B2	SUPDTE	E8DE
UPDTP	E7FC						

```
002                             ORG   :EA0B
003                     *
004                     *
005                     *
006                     **************
007                     * FILL BLOCK *
008                     **************
009                     *
010                     * Fills a rectangular block of whole fields with
011                     * one colour.
012                     *
013                     * Entry: HL:    Address bottom left corner.
014                     *        DE:    Y,X-counts of size of block (E in
015                     *               fields).
016                     *        FCOLR: Colour info.
017                     * Exit:  BCDEHL preserved. AF corrupted.
018                     *
019 EA0B C5             FILBK   PUSH  B
020 EA0C D5                     PUSH  D
021 EA0D E5                     PUSH  H
022 EA0E 3AC200                 LDA   :00C2     ) Get details for colour
023 EA11 4F                     MOV   C,A       ) required in BC
024 EA12 3AC300                 LDA   :00C3     )
025 EA15 47                     MOV   B,A       )
026 EA16 1C                     INR   E         Count range up by 1
027 EA17 D5             FBK10   PUSH  D
028 EA18 E5                     PUSH  H
029 EA19 3AC100         FBK20   LDA   :00C1     Get animate flag
030 EA1C B7                     ORA   A
031 EA1D C246EA                 JNZ   :EA46     Jump if set
032 EA20 3A9D00                 LDA   :009D     Get current screen mode
033 EA23 1F                     RAR
034 EA24 1F                     RAR
035 EA25 70                     MOV   M,B       Colour details in screen RAM
036 EA26 2B                     DCX   H
037 EA27 D23EEA                 JNC   :EA3E     Jump if 16-colour mode
038
039                     * If 4-colour mode:
040
041 EA2A 71                     MOV   M,C       Colour details in screen RAM
042                     *
043 EA2B 2B             FBK30   DCX   H
044 EA2C 1D                     DCR   E
045 EA2D C219EA                 JNZ   :EA19     Loop to do all fields
046 EA30 E1                     POP   H         HL pnts to left of rectangle
047 EA31 D1                     POP   D         Get Y-size count
048 EA32 15                     DCR   D
049 EA33 14                     INR   D
050 EA34 CA53EA                 JZ    :EA53     Abort if ready
051 EA37 15                     DCR   D         Update Y-count
052 EA38 CDC1EA                 CALL  :EAC1     Update pntr to next line
053 EA3B C317EA                 JMP   :EA17     Next line
054
055                     * If 16-colour mode:
056
057 EA3E 7E             FBK40   MOV   A,M       Get data from screen RAM
058 EA3F E60F                   ANI   :0F       Lonibble only (old BG)
059 EA41 B1                     ORA   C         Add details
060 EA42 77                     MOV   M,A       Preserve old background
061 EA43 C32BEA                 JMP   :EA2B
062
063                     * If animate:
```

```
064
065 EA46 E5          FBK50   PUSH  H
066 EA47 05                  DCR   B
067 EA48 04                  INR   B
068 EA49 C24DEA              JNZ   :EA4D
069 EA4C 2B                  DCX   H
070 EA4D 71          FBK60   MOV   M,C       Change whole field
071 EA4E E1                  POP   H
072 EA4F 2B                  DCX   H
073 EA50 C32BEA              JMP   :EA2B     Next field
074                  *
075 EA53 E1          FBK90   POP   H
076 EA54 D1                  POP   D
077 EA55 C1                  POP   B
078 EA56 C9                  RET
079                  *
080                  ***************
081                  * FILL STRIP *
082                  ***************
083                  *
084                  * Fills a vertical strip on the screen with one
085                  * colour.
086                  *
087                  * Entry: HL: Points to bottom field of strip.
088                  *         B:  Mask bits to change.
089                  *         D:  Heigth -1 of strip.
090                  * Exit:  AF currupted, BCDEHL preserved.
091                  *
092 EA57 C5          FILST   PUSH  B
093 EA58 D5                  PUSH  D
094 EA59 E5                  PUSH  H
095 EA5A 3A9D00              LDA   :009D     Get current screen mode
096 EA5D 1F                  RAR
097 EA5E 1F                  RAR
098 EA5F D2A9EA              JNC   :EAA9     Jump if 16-colour mode
099
100                  * If 4-colour mode:
101
102 EA62 D5                  PUSH  D
103 EA63 EB                  XCHG
104 EA64 2AC200              LHLD  :00C2     Get details for colour reqd
105 EA67 3AC100              LDA   :00C1     Get animate flag
106 EA6A B7                  ORA   A
107 EA6B C291EA              JNZ   :EA91     Jump if set
108 EA6E 78                  MOV   A,B       )
109 EA6F A4                  ANA   H         )
110 EA70 4F                  MOV   C,A       ) Mask for bits to be update
111 EA71 78                  MOV   A,B       )
112 EA72 A5                  ANA   L         )
113 EA73 6F                  MOV   L,A       )
114 EA74 78                  MOV   A,B
115 EA75 2F                  CMA             Bits to be preserved
116 EA76 47                  MOV   B,A
117 EA77 67                  MOV   H,A
118 EA78 EB          FST05   XCHG
119 EA79 78          FST10   MOV   A,B
120 EA7A A6                  ANA   M         Pick up old colours
121 EA7B B1                  ORA   C
122 EA7C 77                  MOV   M,A       Update top bits
123 EA7D 2B                  DCX   H
124 EA7E 7A                  MOV   A,D
125 EA7F A6                  ANA   M
```

```
126 EA80 B3              ORA   E
127 EA81 77              MOV   M,A       Update bottom bits
128 EA82 23              INX   H
129 EA83 E3              XTHL
130 EA84 7C              MOV   A,H
131 EA85 25              DCR   H
132 EA86 B7              ORA   A
133 EA87 CABCEA          JZ    :EABC     Jump if ready
134 EA8A E3              XTHL
135 EA8B CDC1EA          CALL  :EAC1     Update pointer
136 EA8E C379EA          JMP   :EA79     Next line
137
138                * If animate:
139
140 EA91 E5      FST15   PUSH  H         Preserve colour details
141 EA92 78              MOV   A,B
142 EA93 A5              ANA   L
143 EA94 4F              MOV   C,A       Bits to be set in C
144 EA95 78              MOV   A,B
145 EA96 2F              CMA
146 EA97 B5              ORA   L
147 EA98 47              MOV   B,A       To be set
148 EA99 2E00            MVI   L,:00
149 EA9B 26FF            MVI   H,:FF     For other byte
150 EA9D F1              POP   PSW       Get colour details
151 EA9E B7              ORA   A
152 EA9F C2A6EA          JNZ   :EAA6
153 EAA2 C5              PUSH  B
154 EAA3 E5              PUSH  H
155 EAA4 C1              POP   B
156 EAA5 E1              POP   H
157 EAA6 C378EA  FST18   JMP   :EA78
158
159                * If 16-colour mode:
160
161 EAA9 3AC200  FST20   LDA   :00C2     Get 1 byte of details colour
162                                      required
163 EAAC 4F              MOV   C,A       Set colour as reqd
164 EAAD CDDEE8  FST30   CALL  :E8DE     Update
165 EAB0 7A              MOV   A,D
166 EAB1 15              DCR   D
167 EAB2 B7              ORA   A
168 EAB3 CABDEA          JZ    :EABD
169 EAB6 CDC1EA          CALL  :EAC1     Next line
170 EAB9 C3ADEA          JMP   :EAAD     Next field
171
172                * If ready:
173
174 EABC E1      FST90   POP   H
175 EABD E1      FST91   POP   H
176 EABE D1              POP   D
177 EABF C1              POP   B
178 EAC0 C9              RET
179                *
180                ****************************
181                * MOVE AND CHECK POINTER *
182                ****************************
183                *
184                * DADCK: Moves a pointer 1 line up screen and
185                *        offsets to alternate area if necessary.
186                * PTRCK: Checks a memory pointer and moves it
187                *        into or out of the archive area if
```

```
188                     *       necessary.
189                     *
190                     * Exit:  HL: New pointer.
191                     *
192 EAC1 3A9800         DADCK   LDA   :0098     Get nr bytes/line
193 EAC4 CD01E7                 CALL  :E701     Add 1 line length
194 EAC7 C5             PTRCK   PUSH  B
195 EAC8 D5                     PUSH  D
196 EAC9 EB                     XCHG
197 EACA 2A8800                 LHLD  :0088     Get addr after end graphics
198                                             area
199 EACD CDFBE6                 CALL  :E6FB     Compare HL-DE
200 EAD0 2A8400                 LHLD  :0084     Get bottom archive area
201 EAD3 44                     MOV   B,H       ) in BC
202 EAD4 4D                     MOV   C,L       )
203 EAD5 2A8200                 LHLD  :0082     Get top visible area
204 EAD8 D2E5EA                 JNC   :EAE5     Jump if pntr is below
205                                             visible screen
206 EADB CDFBE6                 CALL  :E6FB     Compare HL-DE
207 EADE DA0FD8                 JC    :D80F     Jump if pntr is off top
208                                             visible screen
209 EAE1 EB             PCK10   XCHG
210 EAE2 D1             PCK15   POP   D
211 EAE3 C1                     POP   B
212 EAE4 C9                     RET
213
214                     * If pntr is below visible screen:
215
216 EAE5 E5             PCK20   PUSH  H         )
217 EAE6 C5                     PUSH  B         ) Swop BC and HL
218 EAE7 E1                     POP   H         )
219 EAE8 C1                     POP   B         )
220 EAE9 CDFBE6                 CALL  :E6FB     Compare HL-DE
221 EAEC DAE1EA                 JC    :EAE1     Jump if within archive area
222 EAEF EB             PCK30   XCHG
223 EAF0 CDF2E6                 CALL  :E6F2     Subtract nearest boundary
224 EAF3 09                     DAD   B         Add other
225 EAF4 C3E2EA                 JMP   :EAE2
226                     *
227                     ****************************
228                     * FILL A RECTANGULAR AREA *
229                     ****************************
230                     *
231                     * Entry: DE: Width.
232                     *        B : Height.
233                     *        C : Offset.
234                     *        HL: Address.
235                     * Exit: All registers preserved.
236                     *
237 EAF7 F5             FILRT   PUSH  PSW
238 EAF8 C5                     PUSH  B
239 EAF9 D5                     PUSH  D
240 EAFA E5                     PUSH  H
241 EAFB E5                     PUSH  H
242 EAFC 79                     MOV   A,C
243 EAFD 83                     ADD   E
244 EAFE 67                     MOV   H,A       H = C + E
245 EAFF 3E00                   MVI   A,:00
246 EB01 8A                     ADC   D
247 EB02 50                     MOV   D,B
248 EB03 1F                     RAR
249 EB04 7C                     MOV   A,H
```

```
250 EB05 DA0DEB             JC    :EB0D
251 EB08 FE08               CPI   :08
252 EB0A DA40EB             JC    :EB40     If > 8: Fill only one strip
253 EB0D 1F         L2E152  RAR
254 EB0E 0F                 RRC
255 EB0F E67E               ANI   :7E
256 EB11 E3                 XTHL
257 EB12 F5                 PUSH  PSW
258 EB13 0F                 RRC
259 EB14 D602               SUI   :02
260 EB16 5F                 MOV   E,A
261 EB17 2B                 DCX   H
262 EB18 2B                 DCX   H
263 EB19 D40BEA             CNC   :EA0B     Fill block
264 EB1C 23                 INX   H
265 EB1D 23                 INX   H
266 EB1E 79                 MOV   A,C
267 EB1F D609               SUI   :09
268 EB21 2F                 CMA
269 EB22 47                 MOV   B,A
270 EB23 CDE1EB             CALL  :EBE1     Set mask for bits
271 EB26 CD57EA             CALL  :EA57     Fill strip
272 EB29 F1                 POP   PSW
273 EB2A 2F                 CMA
274 EB2B 4F                 MOV   C,A
275 EB2C 06FF               MVI   B,:FF
276 EB2E 03                 INX   B
277 EB2F 09                 DAD   B
278 EB30 F1                 POP   PSW
279 EB31 E607               ANI   :07
280 EB33 3C                 INR   A
281 EB34 47                 MOV   B,A
282 EB35 0E00               MVI   C,:00
283 EB37 CDE1EB     L2E153  CALL  :EBE1     Set mask for bits
284 EB3A CD57EA             CALL  :EA57     Fill strip
285 EB3D C338E1             JMP   :E138     Popall, ret
286
287                 * If < 1 field:
288
289 EB40 43         L2E154  MOV   B,E
290 EB41 04                 INR   B
291 EB42 E1                 POP   H
292 EB43 C337EB             JMP   :EB37     Fill a strip only
293                 *
294                 ***************************
295                 * CALCULATE HL = A * HL *
296                 ***************************
297                 *
298                 * Exit: AFBCDE preserved.
299                 *
300 EB46 F5         HLMUL   PUSH  PSW
301 EB47 D5                 PUSH  D
302 EB48 EB                 XCHG            Original HL in DE
303 EB49 210000             LXI   H,:0000   Init result
304 EB4C B7         L2E156  ORA   A
305 EB4D CA5DEB             JZ    :EB5D     Abort if ready
306 EB50 1F                 RAR             )
307 EB51 F5                 PUSH  PSW       )
308 EB52 D256EB             JNC   :EB56     ) Calc HL = A * HL
309 EB55 19                 DAD   D         )
310 EB56 EB         L2E157  XCHG            )
311 EB57 29                 DAD   H         )
```

```
312 EB58 EB                     XCHG            )
313 EB59 F1                     POP   PSW       )
314 EB5A C34CEB                 JMP   :EB4C     Cont multiplication
315 EB5D D1             L2E158  POP   D
316 EB5E F1                     POP   PSW
317 EB5F C9                     RET
318                     *
319                     ***************************
320                     * CALCULATE HL = HL / A *
321                     ***************************
322                     *
323                     * Exit: AFBCDE preserved.
324                     *
325 EB60 F5             HLDIV   PUSH  PSW
326 EB61 B7                     ORA   A
327 EB62 CA78EB                 JZ    :EB78     Abort if A=0
328 EB65 C5                     PUSH  B
329 EB66 D5                     PUSH  D
330 EB67 01FFFF                 LXI   B,:FFFF
331 EB6A 2F                     CMA
332 EB6B 5F                     MOV   E,A
333 EB6C 16FF                   MVI   D,:FF
334 EB6E 13                     INX   D
335 EB6F 19             L2E160  DAD   D
336 EB70 03                     INX   B
337 EB71 DA6FEB                 JC    :EB6F
338 EB74 69                     MOV   L,C       ) Result in HL
339 EB75 60                     MOV   H,B       )
340 EB76 D1                     POP   D
341 EB77 C1                     POP   B
342 EB78 F1             L2E161  POP   PSW
343 EB79 C9                     RET
344                     *
345                     ************************************************
346                     * CHECK IF SUFFICIENT ROOM FOR GRAPHIC MODE *
347                     ************************************************
348                     *
349                     * Exit: CY=1: Insufficient room for mode.
350                     *       CY=0: O.K.
351                     *       ABCDEHL preserved.
352                     *
353 EB7A C5              TPOSN   PUSH  B
354 EB7B F5                     PUSH  PSW
355 EB7C D5                     PUSH  D
356 EB7D 3A9D00                 LDA   :009D     Get current screen mode
357 EB80 C601                   ADI   :01
358 EB82 DA96EB                 JC    :EB96     If mode 0: Abort CY=1
359 EB85 EB                     XCHG
360 EB86 2A9400                 LHLD  :0094     Get nr of hor. blobs
361 EB89 2B                     DCX   H         minus 1
362 EB8A CDFBE6                 CALL  :E6FB     Compare HL-DE
363 EB8D EB                     XCHG
364 EB8E DA96EB                 JC    :EB96     Abort CY=1 if insufficient
365                                             room
366 EB91 3A9600                 LDA   :0096     Get nr of graphics lines
367 EB94 3D                     DCR   A         minus 1
368 EB95 B9                     CMP   C         Set flags on difference
369 EB96 D1             L2E163  POP   D
370 EB97 C1                     POP   B
371 EB98 78                     MOV   A,B
372 EB99 C1                     POP   B
373 EB9A C9                     RET
```

```
                        *
                        ************************************
                        * FIND COLOUR IN COLORG REGISTERS *
                        ************************************
                        *
                        * Entry: C: Requested colour.
                        * Exit:  CY=1: 'Serialnr' of colour in A.
                        *        CY=0: Colour not available.
                        *        BCDEHL preserved.
                        *
EB9B C5         STR164  PUSH  B
EB9C E5                 PUSH  H
EB9D 219E00             LXI   H,:009E    Addr 1st colour byte graph
EBA0 0603               MVI   B,:03      Total 4 colour data
EBA2 7E         L2E165  MOV   A,M        Get colour
EBA3 E60F               ANI   :0F        Colour nibble only
EBA5 B9                 CMP   C          Ident to reqd one ?
EBA6 CAB2EB             JZ    :EBB2      Then colour found
EBA9 23                 INX   H          Pnts to next one
EBAA 05                 DCR   B
EBAB F2A2EB             JP    :EBA2      Get next colour
EBAE B7                 ORA   A          CY=0
EBAF E1         L2E166  POP   H
EBB0 C1                 POP   B
EBB1 C9                 RET

                        * If colour found:

EBB2 3E03       L2E167  MVI   A,:03
EBB4 90                 SUB   B          Colour'nr' in A
EBB5 37                 STC
EBB6 C3AFEB             JMP   :EBAF      Quit, CY=1
                        *
                        ************************
                        * GET MEMORY POINTER *
                        ************************
                        *
EBB9 D5         SMEMMK  PUSH  D
EBBA F5                 PUSH  PSW
EBBB 78                 MOV   A,B
EBBC 0F                 RRC
EBBD 79                 MOV   A,C
EBBE 1F                 RAR
EBBF 1F                 RAR
EBC0 E67E               ANI   :7E
EBC2 C604               ADI   :04
EBC4 5F                 MOV   E,A
EBC5 1600               MVI   D,:00
EBC7 E1                 POP   H
EBC8 E5                 PUSH  H
EBC9 6C                 MOV   L,H        Entry A in L
EBCA 2600               MVI   H,:00
EBCC 23                 INX   H          +1
EBCD 3A9800             LDA   :0098      Get nr bytes/line
EBD0 CD46EB             CALL  :EB46      Calc HL=A*HL
EBD3 CDF2E6             CALL  :E6F2      HL=HL-DE
EBD6 EB                 XCHG             Result in DE
EBD7 2A8800             LHLD  :0088      Get addr after end graph
                                         area
EBDA 19                 DAD   D          Add offset
EBDB CDC7EA             CALL  :EAC7      Check and move pntr
EBDE F1                 POP   PSW
```

```
436 EBDF D1                 POP   D
437 EBE0 C9                 RET
438                 *
439                 **********************
440                 * SET MASK FOR BITS *
441                 **********************
442                 *
443                 * Entry: B and C are data for mask.
444                 * Exit:  B: Result.
445                 *        AFCDEHL preserved.
446                 *
447 EBE1 F5           SMKMSK PUSH  PSW
448 EBE2 C5                  PUSH  B
449 EBE3 AF                  XRA   A          A=0
450 EBE4 37           L2E170 STC               CY=1
451 EBE5 1F                  RAR
452 EBE6 05                  DCR   B
453 EBE7 C2E4EB              JNZ   :EBE4      RAR (B) times
454 EBEA 1F           L2E171 RAR
455 EBEB 0D                  DCR   C
456 EBEC F2EAEB              JP    :EBEA      RAR until C <= 7F
457 EBEF 17                  RAL
458 EBF0 C1                  POP   B
459 EBF1 47                  MOV   B,A        Result in B
460 EBF2 F1                  POP   PSW
461 EBF3 C9                  RET
462                 *
463                 *
464                 *
465 EBF4                     END
```

****
* S Y M B O L T A B L E *
****

DADCK	EAC1	FBK10	EA17	FBK20	EA19	FBK30	EA2B
FBK40	EA3E	FBK50	EA46	FBK60	EA4D	FBK90	EA53
FILBK	EA0B	FILRT	EAF7	FILST	EA57	FST05	EA78
FST10	EA79	FST15	EA91	FST18	EAA6	FST20	EAA9
FST30	EAAD	FST90	EABC	FST91	EABD	HLDIV	EB60
HLMUL	EB46	L2E152	EB0D	L2E153	EB37	L2E154	EB40
L2E156	EB4C	L2E157	EB56	L2E158	EB5D	L2E160	EB6F
L2E161	EB78	L2E163	EB96	L2E165	EBA2	L2E166	EBAF
L2E167	EBB2	L2E170	EBE4	L2E171	EBEA	PCK10	EAE1
PCK15	EAE2	PCK20	EAE5	PCK30	EAEF	PTRCK	EAC7
SMEMMK	EBB9	SMKMSK	EBE1	STR164	EB9B	TPOSN	EB7A

```
002                             ORG   :EBF4
003                     *
004                     *
005                     *
006                     *   ====================
007                     *** SCREEN EDITOR PACKAGE ***
008                     *   ====================
009                     *
010                     *
011                     **********************
012                     * INITIALISE EDITOR *
013                     **********************
014                     *
015                     * Sets up a mode 0 screen, clears 00A8-00AD, 00AF.
016                     * Prints text from 00A2 onwards; cursor at top left.
017                     *
018                     * Exit: HL: Cursor position (top left).
019                     *       AF corrupted, BCDE preserved.
020                     *
021 EBF4 3EFF           EINIT   MVI   A,:FF     Init mode 0
022 EBF6 CDD9E3                CALL  :E3D9     Change to mode 0
023 EBF9 3E0C                   MVI   A,:0C
024 EBFB CD02E1                CALL  :E102     Clear screen
025 EBFE 210000                LXI   H,:0000   )
026 EC01 22A900                SHLD  :00A9     )
027 EC04 22AC00                SHLD  :00AC     )
028 EC07 7C                    MOV   A,H       ) Clear edit pointers
029 EC08 32A800                STA   :00A8     )
030 EC0B 32AB00                STA   :00AB     )
031 EC0E 32AF00                STA   :00AF     )
032 EC11 2A7200                LHLD  :0072     Get cursor pos addr
033 EC14 CD6BE3                CALL  :E36B     Delete cursor
034 EC17 CD3AEF                CALL  :EF3A     Print full screen from
035                                            start of edit buffer
036 EC1A CD30E3                CALL  :E330     Put cursor on screen
037 EC1D C9                    RET
038                     *
039                     *************
040                     * OBEY EDIT *
041                     *************
042                     *
043                     * A given character is inserted in the buffer,
044                     * except:  #08  deletes a character
045                     *          #10  moves cursor up
046                     *          #11  moves cursor down
047                     *          #12  moves cursor left
048                     *          #13  moves cursor right
049                     *          #14  moves window up
050                     *          #15  moves window down
051                     *          #16  moves window left
052                     *          #17  moves window right
053                     *
054                     * Entry: A:    Character given.
055                     * Exit:  CY=1: If done.
056                     *        CY=0: Not done.
057                     *        ABCDEHL + rest of F preserved.
058                     *
059 EC1E FE08           EOBEY   CPI   :08
060 EC20 CACCEF                JZ    :EFCC     Delete character
061 EC23 FE10                  CPI   :10
062 EC25 DA4BEF                JC    :EF4B     Insert character
063 EC28 CA88ED                JZ    :ED88     Cursor up
```

```
064 EC2B FE12              CPI   :12
065 EC2D DAABED            JC    :EDAB     Cursor down
066 EC30 CAD3ED            JZ    :EDD3     Cursor left
067 EC33 FE14              CPI   :14
068 EC35 DAF6ED            JC    :EDF6     Cursor right
069 EC38 CA4BEC            JZ    :EC4B     Window up
070 EC3B FE16              CPI   :16
071 EC3D DAB3EC            JC    :ECB3     Window down
072 EC40 CA50ED            JZ    :ED50     Window left
073 EC43 FE17              CPI   :17
074 EC45 CAF8EC            JZ    :ECF8     Window right
075 EC48 C34BEF            JMP   :EF4B     Insert character
076                  *
077                  *************
078                  * WINDOW UP *
079                  *************
080                  *
081                  * Moves window up one line.
082                  *
083                  * Exit: All registers preserved.
084                  *       CY=0: Window unchanged.
085                  *       CY=1: Window changed.
086                  *
087 EC4B D5          EWUP    PUSH  D
088 EC4C E5                  PUSH  H
089 EC4D F5                  PUSH  PSW
090 EC4E 2AA900              LHLD  :00A9     Get offset of top of window
091 EC51 7C                  MOV   A,H
092 EC52 B5                  ORA   L
093 EC53 CADAEC              JZ    :ECDA     Window unchanged if
094                                          offset = 0; CY=0
095 EC56 AF                  XRA   A
096 EC57 32AF00              STA   :00AF     Clear cursor pos in buffer
097 EC5A 3AAC00              LDA   :00AC     Get Y-offset cursor in
098                                          in document
099 EC5D 95                  SUB   L         Minus offset top of window
100 EC5E FE17                CPI   :17       Nr of lines in window -1
101 EC60 CC88ED              CZ    :ED88     Cursor up if at
102                                          bottom of window
103 EC63 2A7200              LHLD  :0072     Get cursor pos addr
104 EC66 E5                  PUSH  H
105 EC67 CD6BE3              CALL  :E36B     Delete cursor
106 EC6A CD74EC              CALL  :EC74     Move window, correct pntrs
107 EC6D E1                  POP   H
108 EC6E 117AFF    L2E172    LXI   D,:FF7A   Length one line
109 EC71 C3D5EC              JMP   :ECD5     Put cursor on next line,
110                                          quit with CY=1
111                  *
112                  * SCROLL EDIT DISPLAY DOWN ONE LINE:
113                  *
114                  * Exit: BC preserved. AFDEHL corrupted.
115                  *
116 EC74 C5          L2E173  PUSH  B
117 EC75 018600              LXI   B,:0086   Length one screen line
118 EC78 CD86EC              CALL  :EC86     Move window up in text
119                                          1 line
120 EC7B D5                  PUSH  D
121 EC7C 2AA900              LHLD  :00A9     Get offset of top of window
122 EC7F 2B                  DCX   H         -1
123 EC80 22A900              SHLD  :00A9     And preserve it
124 EC83 C3EFEC              JMP   :ECEF     Print new line of text at
125                                          window bottom
```

```
126                     *
127                     ************************
128                     * MOVE WINDOW IN TEXT *
129                     ************************
130                     *
131                     * Moves window up or left.
132                     *
133                     * Entry: BC: Offset (#86 for 1 line, #02 for
134                     *            1 position).
135                     * Exit:  All registers corrupted.
136                     *
137 EC86 2A8C00         L2E174  LHLD  :008C     Get addr after end char area
138 EC89 54                     MOV   D,H       ) in DE
139 EC8A 5D                     MOV   E,L       )
140 EC8B 09                     DAD   B         Add offset
141 EC8C EB                     XCHG            HL: end char area;
142                                             DE: end char area + offset
143                     * Move full screen:
144
145 EC8D 010600         L2E175  LXI   B,:0006   3 not used locations
146                                             at line end
147 EC90 09                     DAD   B         )
148 EC91 EB                     XCHG            ) Add it to both HL and DE
149 EC92 09                     DAD   B         ) and exchange DE and HL
150 EC93 EB                     XCHG            )
151 EC94 063C                   MVI   B,:3C     60 char/line visible
152
153                     * Move one line:
154
155 EC96 23              L2E176  INX   H
156 EC97 23                     INX   H         New destination pntr
157 EC98 13                     INX   D
158 EC99 13                     INX   D         New origin pntr
159 EC9A 1A                     LDAX  D         Get one char
160 EC9B 77                     MOV   M,A       Move it to new screen loc
161 EC9C 05                     DCR   B         char count -1
162 EC9D C296EC                 JNZ   :EC96     Next char if not ready
163                     *
164 ECA0 010800                 LXI   B,:0008   4 not-useable pos at line
165                                             boundary
166 ECA3 09                     DAD   B         HL pnts to end previous line
167                                             (destination)
168 ECA4 E5                     PUSH  H
169 ECA5 2A8A00                 LHLD  :008A     Get addr start char area
170 ECA8 EB                     XCHG            in DE
171 ECA9 09                     DAD   B         Update origin too
172 ECAA CDFBE6                 CALL  :E6FB     Compare HL-DE
173 ECAD EB                     XCHG            New origin in DE
174 ECAE E1                     POP   H         Destination in HL
175 ECAF DA8DEC                 JC    :EC8D     Evt another line
176 ECB2 C9                     RET
177                     *
178                     ***************
179                     * WINDOW DOWN *
180                     ***************
181                     *
182                     * Moves window down one line.
183                     *
184                     * Exit: ABCDEHL preserved.
185                     *       CY=0: Window unchanged.
186                     *       CY=1: Window changed.
187                     *
```

```
188 ECB3 D5          EWDN    PUSH  D
189 ECB4 E5                  PUSH  H
190 ECB5 F5                  PUSH  PSW
191 ECB6 2AAC00              LHLD  :00AC     Get Y-offset cursor in
192                                          document
193 ECB9 3AA900              LDA   :00A9     Get offset of top of window
194 ECBC BD                  CMP   L         Cursor at top of screen ?
195 ECBD CCABED              CZ    :EDAB     Then cursor down
196 ECC0 D2DAEC              JNC   :ECDA     Quit with CY=0 if
197                                          window unchanged
198 ECC3 AF                  XRA   A
199 ECC4 32AF00              STA   :00AF     Clear cursor pos in buffer
200 ECC7 2A7200              LHLD  :0072     Get cursor pos addr
201 ECCA E5                  PUSH  H
202 ECCB CD6BE3              CALL  :E36B     Delete cursor
203 ECCE CDDFEC              CALL  :ECDF     Scroll edit display up
204                                          one line
205 ECD1 E1                  POP   H
206 ECD2 118600      L2E177  LXI   D,:0086   Length one screen line
207 ECD5 19          L2E178  DAD   D         One line further
208 ECD6 CD30E3      L2E179  CALL  :E330     Put new cursor on screen
209 ECD9 37                  STC             Exit CY=1 if cursor moved
210 ECDA E1          L2E180  POP   H
211 ECDB 7C                  MOV   A,H
212 ECDC E1                  POP   H
213 ECDD D1                  POP   D
214 ECDE C9                  RET
215                  *
216                  * SCROLL EDIT DISPLAY UP ONE LINE:
217                  *
218                  * Exit: BC preserved, AFDEHL corrupted.
219                  *
220 ECDF C5          L2E181  PUSH  B
221 ECE0 CDCBE1              CALL  :E1CB     Scroll window up 1 line
222 ECE3 E5                  PUSH  H
223 ECE4 2AA900              LHLD  :00A9     Get offset of top of window
224 ECE7 23                  INX   H         +1
225 ECE8 22A900              SHLD  :00A9     Preserve it
226 ECEB 111700              LXI   D,:0017   Nr of lines for window
227 ECEE 19                  DAD   D         HL pnts to new line at
228                                          window bottom
229 ECEF CD1CEE      L2E182  CALL  :EE1C     Skip lines
230 ECF2 D1                  POP   D
231 ECF3 CDC0EE              CALL  :EEC0     Print new line of text
232 ECF6 C1                  POP   B
233 ECF7 C9                  RET
234                  *
235                  ****************
236                  * WINDOW RIGHT *
237                  ****************
238                  *
239                  * Moves window right one position.
240                  *
241                  * Exit: ABCDEHL preserved.
242                  *       CY=0: Window unchanged.
243                  *       CY=1: Window changed.
244                  *
245 ECF8 D5          EWRT    PUSH  D
246 ECF9 E5                  PUSH  H
247 ECFA F5                  PUSH  PSW
248 ECFB 3AA800              LDA   :00A8     Get offset of left side
249                                          of window
```

```
250 ECFE FEC4               CPI   :C4       256-60 ?
251 ED00 CADAEC             JZ    :ECDA     Abort if window at right en
252                                           of text
253 ED03 6F                 MOV   L,A       Offset in L
254 ED04 3AAB00             LDA   :00AB     Get X-offset cursor in
255                                           document
256 ED07 BD                 CMP   L         Cursor at left side of
257                                           window ?
258 ED08 CCF6ED             CZ    :EDF6     Then move cursor right
259 ED0B 2A7200             LHLD  :0072     Get cursor pos addr
260 ED0E E5                 PUSH  H
261 ED0F CD6BE3             CALL  :E36B     Delete cursor
262 ED12 CD1BED             CALL  :ED1B     Scroll display left 1 pos
263 ED15 E1                 POP   H
264 ED16 23          L2E184 INX   H
265 ED17 23                 INX   H
266 ED18 C3D6EC             JMP   :ECD6     Put cursor on screen;
267                                           quit with CY=1
268                 *
269                 * SCROLL EDIT DISPAY LEFT ONE POSITION:
270                 *
271                 * Exit: BC preserved, AFDEHL corrupted.
272                 *
273 ED1B C5          L2E185 PUSH  B
274 ED1C 01FEFF             LXI   B,:FFFE   -2
275 ED1F CDCEE1             CALL  :E1CE     Scroll window
276 ED22 3AA800             LDA   :00A8     Get offset of left side
277                                           of window
278 ED25 3C                 INR   A         +1
279 ED26 32A800             STA   :00A8     Preserve it
280 ED29 C63B               ADI   :3B       +60 (offset right side
281                                           of window)
282 ED2B 1182FF             LXI   D,:FF82   -7E
283 ED2E 47          L2E186 MOV   B,A       Offset right side of
284                                           window in B
285 ED2F 0E18               MVI   C,:18     Nr of lines in window
286 ED31 2A8A00             LHLD  :008A     Get pntr to start char area
287 ED34 19                 DAD   D         Pos 60th char on screen
288 ED35 EB                 XCHG            in DE
289 ED36 2AA900             LHLD  :00A9     Get offset of top of window
290 ED39 CD1CEE             CALL  :EE1C     Skip lines
291
292                 * Put newly visible 60th character on screen:
293
294 ED3C CD7BEE      L2E187 CALL  :EE7B     Skip to Bth pos on line
295 ED3F 12                 STAX  D         Next visible char in window
296 ED40 CD38EE             CALL  :EE38     Next line
297 ED43 E5                 PUSH  H
298 ED44 217AFF             LXI   H,:FF7A   -86 (length one line)
299 ED47 19                 DAD   D         Pnts to 60th char on
300                                           next line
301 ED48 EB                 XCHG
302 ED49 E1                 POP   H
303 ED4A 0D                 DCR   C         Update line count
304 ED4B C23CED      L2E188 JNZ   :ED3C     Scroll next line if not
305                                           ready
306 ED4E C1                 POP   B
307 ED4F C9                 RET
308                 *
309                 ***************
310                 * WINDOW LEFT *
311                 ***************
```

```
312                     *
313                     * Moves window left one position.
314                     *
315                     * Exit:  ABCDEHL preserved.
316                     *        CY=0: Window unchanged.
317                     *        CY=1: Window changed.
318                     *
319 ED50 D5             EWLF    PUSH  D
320 ED51 E5                     PUSH  H
321 ED52 F5                     PUSH  PSW
322 ED53 3AA800                 LDA   :00A8     Get offset of left side
323                                             of window
324 ED56 B7                     ORA   A
325 ED57 CADAEC                 JZ    :ECDA     Abort if offset = 0
326 ED5A 6F                     MOV   L,A       Offset in L
327 ED5B 3AAB00                 LDA   :00AB     Get X-offset of cursor
328                                             in document
329 ED5E 95                     SUB   L
330 ED5F FE3B                   CPI   :3B       Cursor at right side of
331                                             window ?
332 ED61 CCD3ED                 CZ    :EDD3     Then cursor left
333 ED64 2A7200                 LHLD  :0072     Get cursor pos addr
334 ED67 E5                     PUSH  H
335 ED68 CD6BE3                 CALL  :E36B     Delete cursor
336 ED6B CD74ED                 CALL  :ED74     Move window, correct pntrs
337 ED6E E1                     POP   H
338 ED6F 2B             L2E189  DCX   H
339 ED70 2B                     DCX   H         New cursor pos
340 ED71 C3D6EC                 JMP   :ECD6     Put cursor on screen;
341                                             quit with CY=1
342                     *
343                     * SCROLL EDIT DISPLAY RIGHT ONE POSITION:
344                     *
345                     * Exit:  BC preserved; AFDEHL corrupted.
346                     *
347 ED74 C5             L2E190  PUSH  B
348 ED75 010200                 LXI   B,:0002
349 ED78 CD86EC                 CALL  :EC86     Move window in text
350 ED7B 3AA800                 LDA   :00A8     Get offset of left side
351                                             of window
352 ED7E 3D                     DCR   A         -1
353 ED7F 32A800                 STA   :00A8     Preserve it
354 ED82 11F8FF                 LXI   D,:FFF8   Gives in ED2E 1st pos
355                                             on 1st screen line
356 ED85 C32EED                 JMP   :ED2E     Scroll display left
357                     *
358                     *************
359                     * CURSOR UP *
360                     *************
361                     *
362                     * Moves cursor up one position.
363                     *
364                     * Exit:  ABCDEHL preserved.
365                     *        CY=0: Cursor not moved.
366                     *        CY=1: Cursor moved.
367                     *
368 ED88 D5             ECUP    PUSH  D
369 ED89 E5                     PUSH  H
370 ED8A F5                     PUSH  PSW
371 ED8B 2AAC00                 LHLD  :00AC     Get Y-offset cursor in
372                                             document
373 ED8E 7C                     MOV   A,H
```

```
374 ED8F B5                 ORA   L
375 ED90 CADAEC             JZ    :ECDA     Quit if cursor at top
376                                           left; CY=0
377 ED93 AF                 XRA   A
378 ED94 32AF00             STA   :00AF     Clear cursor pos in buffer
379 ED97 3AA900             LDA   :00A9     Get offset of top of window
380 ED9A BD                 CMP   L         Cursor at top of window ?
381 ED9B CC4BEC             CZ    :EC4B     Then window up
382 ED9E 2B                 DCX   H
383 ED9F 22AC00             SHLD  :00AC     Store Y-offset cursor in
384                                           document
385 EDA2 2A7200             LHLD  :0072     Get cursor pos addr
386 EDA5 CD6BE3             CALL  :E36B     Delete cursor
387 EDA8 C3D2EC             JMP   :ECD2     Put cursor on new pos;
388                                           quit with CY=1
389                 *.
390                 ***************
391                 * CURSOR DOWN *
392                 ***************
393                 *
394                 * Moves cursor down one position.
395                 *
396                 * Exit: ABCDEHL preserved.
397                 *       CY=0: Cursor not moved.
398                 *       CY=1: Cursor moved.
399                 *
400 EDAB D5        ECDN    PUSH  D
401 EDAC E5                PUSH  H
402 EDAD F5                PUSH  PSW
403 EDAE 2AAC00            LHLD  :00AC     Get Y-offset cursor in
404                                           in document
405 EDB1 23                INX   H         +1
406 EDB2 E5                PUSH  H         Preserve it
407 EDB3 CD1CEE            CALL  :EE1C     Skip lines
408 EDB6 E1                POP   H
409 EDB7 D2DAEC            JNC   :ECDA     No move if cursor at last
410                                           line of text; CY=0
411 EDBA AF                XRA   A
412 EDBB 32AF00            STA   :00AF     Clear cursor pos in buffer
413 EDBE 3AA900            LDA   :00A9     Get offset of top of window
414 EDC1 C618              ADI   :18       24 lines in window
415 EDC3 BD                CMP   L         Cursor at window bottom ?
416 EDC4 CCB3EC            CZ    :ECB3     Then window down
417 EDC7 22AC00            SHLD  :00AC     Store Y-offset cursor in
418                                           document
419 EDCA 2A7200            LHLD  :0072     Get cursor pos addr
420 EDCD CD6BE3            CALL  :E36B     Delete cursor
421 EDD0 C36EEC            JMP   :EC6E     Put cursor on new pos;
422                                           quit with CY=1
423                 *
424                 ***************
425                 * CURSOR LEFT *
426                 ***************
427                 *
428                 * Moves cursor left one position.
429                 *
430                 * Exit: ABCDEHL preserved.
431                 *       CY=0: Cursor not moved.
432                 *       CY=1: Cursor moved.
433                 *
434 EDD3 D5        ECLF    PUSH  D
435 EDD4 E5                PUSH  H
```

```
436 EDD5 F5                     PUSH   PSW
437 EDD6 3AA800                 LDA    :00A8      Get offset of left side
438                                                of window
439 EDD9 6F                     MOV    L,A        in L
440 EDDA 3AAB00                 LDA    :00AB      Get X-offset cursor in
441                                                document
442 EDDD B7                     ORA    A
443 EDDE CADAEC                 JZ     :ECDA      Abort if X-offset = 0
444 EDE1 BD                     CMP    L          Cursor at left side of
445                                                window ?
446 EDE2 CC50ED                 CZ     :ED50      Then window left
447 EDE5 3D                     DCR    A          X-offset -1
448 EDE6 32AB00                 STA    :00AB      Preserve it
449 EDE9 AF                     XRA    A
450 EDEA 32AF00                 STA    :00AF      Clear cursor pos in buffer
451 EDED 2A7200                 LHLD   :0072      Get cursor pos addr
452 EDF0 CD6BE3                 CALL   :E36B      Delete cursor
453 EDF3 C316ED                 JMP    :ED16      Put cursor on new pos;
454                                                quit with CY=1
455                     *
456                     *
457                     *
458 EDF6                        END
```

```
*****************************
* S Y M B O L   T A B L E *
*****************************
```

ECDN	EDAB	ECLF	EDD3	ECUP	ED88	EINIT	EBF4
EOBEY	EC1E	EWDN	ECB3	EWLF	ED50	EWRT	ECF8
EWUP	EC4B	L2E172	EC6E	L2E173	EC74	L2E174	EC86
L2E175	EC8D	L2E176	EC96	L2E177	ECD2	L2E178	ECD5
L2E179	ECD6	L2E180	ECDA	L2E181	ECDF	L2E182	ECEF
L2E184	ED16	L2E185	ED1B	L2E186	ED2E	L2E187	ED3C
L2E188	ED4B	L2E189	ED6F	L2E190	ED74		

```
002                             ORG   :EDF6
003                     *
004                     *
005                     *
006                     ****************
007                     * CURSOR RIGHT *
008                     ****************
009                     *
010                     * Moves cursor right one position.
011                     *
012                     * Exit: ABCDEHL preserved.
013                     *       CY=0: Cursor not moved.
014                     *       CY=1: Cursor moved.
015                     *
016 EDF6 D5             ECRT    PUSH  D
017 EDF7 E5                     PUSH  H
018 EDF8 F5                     PUSH  PSW
019 EDF9 3AA800                 LDA   :00A8     Get offset of left side
020                                             of window
021 EDFC C63B                   ADI   :3B       Calc end of window
022 EDFE 6F                     MOV   L,A       Result in L
023 EDFF 3AAB00                 LDA   :00AB     Get X-offset of cursor
024                                             in document
025 EE02 FEFF                   CPI   :FF       Max nr char/line reached ?
026 EE04 CADAEC                 JZ    :ECDA     Then quit with CY=0
027 EE07 BD                     CMP   L         End of window reached ?
028 EE08 CCF8EC                 CZ    :ECF8     Then window right
029 EE0B 3C                     INR   A         Incr cursor pos on line
030 EE0C 32AB00                 STA   :00AB     Store X-offset of cursor
031                                             in document
032 EE0F AF                     XRA   A
033 EE10 32AF00                 STA   :00AF     Clear cursor pos in buffer
034 EE13 2A7200                 LHLD  :0072     Get cursor pos addr
035 EE16 CD6BE3                 CALL  :E36B     Delete old cursor
036 EE19 C36FED                 JMP   :ED6F     Put new cursor on screen;
037                                             quit with CY=1
038                     *
039                     ******************************
040                     * SKIP LINES IN EDIT BUFFER *
041                     ******************************
042                     *
043                     * Entry: HL: Number of lines to be skipped.
044                     * Exit:  HL: Points to 1st not skipped line.
045                     *        CY=1: O.K.
046                     *        CY=0: End of text reached before count
047                     *              is zero.
048                     *        ABCDE preserved.
049                     *
050 EE1C D5             L2E191  PUSH  D
051 EE1D F5                     PUSH  PSW
052 EE1E EB                     XCHG            Count in DE
053 EE1F 2AA200                 LHLD  :00A2     Get startaddr editbuf
054 EE22 7A             L2E192  MOV   A,D       )
055 EE23 B3                     ORA   E         ) Abort with CY=1 if ready
056 EE24 CA33EE                 JZ    :EE33     )
057 EE27 CD38EE                 CALL  :EE38     Next line
058 EE2A 00                     NOP
059 EE2B B7                     ORA   A
060 EE2C CA34EE                 JZ    :EE34     Abort with CY=0 if end of
061                                             buffer reached before DE=0
062 EE2F 1B                     DCX   D         Count -1
063 EE30 C322EE                 JMP   :EE22     Skip next line
```

```
064 EE33 37          L2E193  STC
065 EE34 D1          L2E194  POP   D
066 EE35 7A                  MOV   A,D
067 EE36 D1                  POP   D
068 EE37 C9                  RET
069                  *
070                  *************
071                  * NEXT LINE *
072                  *************
073                  *
074                  * Entry: HL: points to text, each line ended by
075                  *             a car.ret, last line followed by 0.
076                  * Exit:  HL: Points to next line.
077                  *        A:  0 if end of text reached, else CR.
078                  *        BCDE preserved.
079                  *
080 EE38 7E          L2E195  MOV   A,M       Get char from text
081 EE39 B7                  ORA   A
082 EE3A CA43EE              JZ    :EE43     Ready if end of text reached
083 EE3D 23                  INX   H
084 EE3E FE0D                CPI   :0D       Car.ret ?
085 EE40 C238EE              JNZ   :EE38     Loop till end of line found
086 EE43 C9          L2E196  RET
087                  *
088                  ****************************************
089                  * FIND CURRENT POSITION IN EDIT BUFFER *
090                  ****************************************
091                  *
092                  * Exit: CY=1: HL: (00AF/AE) = cursor position in
093                  *                 buffer.
094                  *       CY=0: HL: Points to cursor line in the
095                  *                 edit buffer.
096                  *       ABCDE preserved.
097                  *
098 EE44 2AAE00      L2E197  LHLD  :00AE     Get pntr to cursor pos
099                                          in buffer
100 EE47 25                  DCR   H
101 EE48 24                  INR   H
102 EE49 37                  STC
103 EE4A C0                  RNZ           Quit (CY=1) if (AF) <> 0
104 EE4B D5                  PUSH  D
105 EE4C C5                  PUSH  B
106 EE4D F5                  PUSH  PSW
107 EE4E 2AAC00              LHLD  :00AC     Get Y-offset of cursor
108                                          in document (no lines to
109                                          be skipped)
110 EE51 54                  MOV   D,H       ) in DE
111 EE52 5D                  MOV   E,L       )
112 EE53 CD1CEE              CALL  :EE1C     Skip lines
113 EE56 22B200              SHLD  :00B2     Store pntr to start cursor
114                                          line in buffer
115 EE59 E5                  PUSH  H         and preserve it
116 EE5A 2AA900              LHLD  :00A9     Get offset of top of window
117 EE5D CDF2E6              CALL  :E6F2     Calc difference
118 EE60 3E86                MVI   A,:86     Length one line
119 EE62 CD46EB              CALL  :EB46     Calc total length
120 EE65 EB                  XCHG            Result in DE
121 EE66 2A8A00              LHLD  :008A     Get startaddr char area
122 EE69 19                  DAD   D         Add offset
123 EE6A 22B000              SHLD  :00B0     Preserve pntr to start
124                                          cursor line on screen
125 EE6D E1                  POP   H         Get pntr to start cursor
```

```
                                                line in buffer
EE6E 3AAB00             LDA   :00AB     Get X-offset of cursor
                                        in document
EE71 47                 MOV   B,A       in B
EE72 CD7BEE             CALL  :EE7B     Skip to Bth pos on line
                                        exit: HL pnts to cursor pos
EE75 C1                 POP   B
EE76 78                 MOV   A,B
EE77 C1                 POP   B
EE78 D1                 POP   D
EE79 3F                 CMC             Abort with CY=0
EE7A C9                 RET
                *
                ***************************************
                * SKIP TO Bth POSITION ON TEXT LINE *
                ***************************************
                *
                * Looks through a textline until Bth position is
                * found. On exit, character on this position is
                * in A.
                *
                * Entry: HL: Start text line.
                *         B:  Number of position in line.
                * Exit:  CY=0: Position found:
                *              A:  Character on that position.
                *              HL: Points to Bth position.
                *              BCDE preserved.
                *        CY=1: Bth position is beyond car.ret,
                *              tab or end of text.
                *              HL: Points to CR, tab or 0.
                *              A:  Space.
                *
EE7B C5         L2E198  PUSH  B
EE7C D5                 PUSH  D
EE7D 0E00               MVI   C,:00     Init count
EE7F C3FDD1     L2E199  JMP   :D1FD     Evaluate character
EE82 CAA3EE     L2E200  JZ    :EEA3     Abort if Bth pos reached
EE85 B7                 ORA   A         Set flags on char
EE86 CAA0EE             JZ    :EEA0     Jump if end of text reached
EE89 FE0D               CPI   :0D
EE8B CAA0EE             JZ    :EEA0     Jump if char is car.ret
EE8E FE09               CPI   :09
EE90 CA98EE             JZ    :EE98     Jump if char is tab
EE93 0C                 INR   C         Incr count
EE94 23         L2E201  INX   H         Pnts to next pos on line
EE95 C37FEE             JMP   :EE7F     Loop untill ready

                * If character is tab:

EE98 CDA6EE     L2E202  CALL  :EEA6     Tabulate
EE9B 78                 MOV   A,B       Reqd pos in A
EE9C B9                 CMP   C         Compare with tab stops
EE9D D294EE             JNC   :EE94     Continu if not past tab

                * If end of line or end of text reached or if
                * past tab:

EEA0 3E20       L2E203  MVI   A,:20     Set char is space
EEA2 37                 STC             Abort with CY=1
EEA3 D1         L2E204  POP   D
EEA4 C1                 POP   B
EEA5 C9                 RET
```

```
                        *
                        ************
                        * TABULATE *
                        ************
                        *
                        * Routine goes through tab-table for 1st
                        * tab-stop > C.
                        *
                        * Entry: 00B4/B5:  Pointer to tab-table.
                        *        C:        Line position.
                        * Exit:  If found: Tab in C.
                        *        Else:     C = C + 1.
                        *        AFBDEHL preserved.
                        *
EEA6 F5         L2E205  PUSH  PSW
EEA7 C5                 PUSH  B
EEA8 E5                 PUSH  H
EEA9 2AB400             LHLD  :00B4      Get addr tab table
EEAC 41         L2E206  MOV   B,C        Line pos in B
EEAD 04                 INR   B          +1
EEAE 7E                 MOV   A,M        Get tab from table
EEAF B7                 ORA   A
EEB0 CABAEE             JZ    :EEBA      If end tab table reached
EEB3 23                 INX   H          Pnts to next tab
EEB4 47                 MOV   B,A        Tab in B
EEB5 79                 MOV   A,C        Line pos in A
EEB6 B8                 CMP   B
EEB7 D2ACEE             JNC   :EEAC      Get next tab if line
                                         pos > tab stop
EEBA 48         L2E207  MOV   C,B        Replace line pos by tab stop
                                         or by line pos +1 if no tab
                                         found
EEBB E1                 POP   H
EEBC F1                 POP   PSW
EEBD 47                 MOV   B,A
EEBE F1                 POP   PSW
EEBF C9                 RET
                        *
                        ********************************
                        * PRINT LINE OF TEXT IN WINDOW *
                        ********************************
                        *
                        * Entry: HL:   Address of textline in buffer.
                        *        DE:   Line control byte of screen line
                        *              to print text on.
                        *        00A8: Number of non-printing positions.
                        *
                        * During execution loop:
                        *        B:    Nr of non-printing pos left +1.
                        *        C:    Nr of screen line pos left.
                        *        D:    Current text line position.
                        *        E:    Count of blanks inserted.
                        *        HL:   Points to text.
                        *        TOS:  Points to screen.
                        *
EEC0 C5         L2E208  PUSH  B
EEC1 EB                 XCHG
EEC2 01F8FF             LXI   B,:FFF8
EEC5 09                 DAD   B          1st printable pos on screen
EEC6 EB                 XCHG             in DE
EEC7 3AA800             LDA   :00A8      Get offset of left side
                                         of window
```

```
250 EECA 3C                 INR   A
251 EECB 47                 MOV   B,A       B is offset +1
252 EECC 0E3C               MVI   C,:3C     60 visible characters
253 EECE D5                 PUSH  D         Preserve start screen line
254 EECF 1600               MVI   D,:00     ) Init current text pos
255 EED1 5A                 MOV   E,D       ) and blanks count
256
257                 * Loop:
258
259 EED2 3E20       L2E209  MVI   A,:20     Space
260 EED4 1D                 DCR   E
261 EED5 1C                 INR   E
262 EED6 C2ECEE             JNZ   :EEEC     E<>0: Update screen pos pntr
263 EED9 7E                 MOV   A,M       Else: Get char from buffer
264 EEDA B7                 ORA   A         Set flags on char
265 EEDB C3D1D1             JMP   :D1D1     Test character
266 EEDE FE0D       L2E210  CPI   :0D
267 EEE0 CAECEE             JZ    :EEEC     Skip tab-handling if CR
268 EEE3 1E00               MVI   E,:00     Reset blanks count
269 EEE5 23                 INX   H         Pnts to next char in text
270 EEE6 FE09               CPI   :09       Tab ?
271 EEE8 CA08EF             JZ    :EF08     Then tab-handling
272 EEEB 1C         L2E211  INR   E
273 EEEC 1D         L2E212  DCR   E         Update blanks count
274 EEED 14                 INR   D         Update line position
275 EEEE 05                 DCR   B         and nr of pos left in window
276 EEEF C2D2EE             JNZ   :EED2     No printing, cont with
277                                         next char
278 EEF2 04                 INR   B
279 EEF3 E3                 XTHL            Screen pos in HL
280 EEF4 77                 MOV   M,A       Display char on screen
281 EEF5 2B                 DCX   H
282 EEF6 2B                 DCX   H         Next screen pos
283 EEF7 E3                 XTHL            Preserve it
284 EEF8 0D                 DCR   C         End of screen line reached ?
285 EEF9 C2D2EE             JNZ   :EED2     Next char if not
286 EEFC CD38EE             CALL  :EE38     Else: next line
287 EEFF E3                 XTHL            HL is screen pos
288 EF00 11FAFF             LXI   D,:FFFA
289 EF03 19                 DAD   D         HL is 1st useable pos
290                                         on next line
291 EF04 EB                 XCHG            into DE
292 EF05 E1                 POP   H
293 EF06 C1                 POP   B
294 EF07 C9                 RET
295
296                 * Tab-handling:
297
298 EF08 C5         L2E213  PUSH  B
299 EF09 4A                 MOV   C,D       C is pos on current line
300 EF0A CDA6EE             CALL  :EEA6     Tabulate
301 EF0D 79                 MOV   A,C       Next tab stop in A
302 EF0E 92                 SUB   D         ) Calc blanks to be inserted
303 EF0F 3D                 DCR   A         )
304 EF10 5F                 MOV   E,A       Update blanks count
305 EF11 C1                 POP   B
306 EF12 3E09               MVI   A,:09     Restore tab char in A
307 EF14 C3EBEE             JMP   :EEEB
308                 *
309                 ****************************
310                 * PRINT A COMPLETE WINDOW *
311                 ****************************
```

```
312                     *
313                     * Text is printed in window from (DE) to bottom
314                     * text screen if text ends. A ASCII 0 (vertical
315                     * bar) is printed at the beginning of all
316                     * following lines.
317                     *
318                     * Entry: HL: Points to text.
319                     *        DE: Points to line control byte screen line
320                     * Exit:  HL: Points to next line to print.
321                     *        DE: Bottom text screen.
322                     *        AF corrupted, BC preserved.
323                     *
324 EF17 E5          L2E214  PUSH  H
325 EF18 2A8C00               LHLD  :008C     Get bottom text screen
326 EF1B CDFBE6             CALL  :E6FB     Compare HL-DE
327 EF1E E1                 POP   H
328 EF1F CA28EF             JZ    :EF28     Quit if window full
329 EF22 CDC0EE             CALL  :EEC0     Print a textline
330 EF25 C317EF             JMP   :EF17     Loop until ready
331 EF28 C9          L2E215  RET
332                     *
333                     ****************************************
334                     * PRINT A TEXT LINE POINTED BY (00B2) *
335                     * ON SCREEN POSITION (00B0) IN WINDOW *
336                     ****************************************
337                     *
338                     * Exit: All registers preserved.
339                     *
340 EF29 F5          L2E216  PUSH  PSW
341 EF2A C5                 PUSH  B
342 EF2B D5                 PUSH  D
343 EF2C E5                 PUSH  H
344 EF2D 2AB000             LHLD  :00B0     Get pntr to start cursor
345                                         line on screen
346 EF30 EB                 XCHG            in DE
347 EF31 2AB200             LHLD  :00B2     Get pntr to start cursor
348                                         line in buffer
349 EF34 CDC0EE             CALL  :EEC0     Print a text line
350 EF37 C338E1             JMP   :E138     Popall, ret
351                     *
352                     **********************
353                     * PRINT FULL SCREEN *
354                     **********************
355                     *
356                     * Prints from Nth line in full screen window.
357                     * N is the offset of the top of the window from
358                     * the start of the edit buffer.
359                     *
360                     * Exit: All registers preserved.
361                     *
362 EF3A F5          L2E217  PUSH  PSW
363 EF3B C5                 PUSH  B
364 EF3C D5                 PUSH  D
365 EF3D E5                 PUSH  H
366 EF3E 2A8A00             LHLD  :008A     Get startaddr char area
367 EF41 EB                 XCHG            in DE
368 EF42 2AA900             LHLD  :00A9     Get offset of top of window
369 EF45 CD1CEE             CALL  :EE1C     Skip lines
370 EF48 C385CE             JMP   :CE85     Print complete window
371                     *
```

```
374                     ********************************
375                     * INSERT CHARACTER IN BUFFER *
376                     ********************************
377                     *
378                     * Entry: Character in A.
379                     * Exit:  ABCDEHL preserved.
380                     *        CY=0: Buffer full, no action.
381                     *        CY=1: Character inserted.
382                     *
383 EF4B C5          EINCH   PUSH  B
384 EF4C F5                  PUSH  PSW
385 EF4D D5                  PUSH  D
386 EF4E E5                  PUSH  H
387 EF4F 47                  MOV   B,A       Char in B
388 EF50 2AA600              LHLD  :00A6     Get end available space
389 EF53 EB                  XCHG            in DE
390 EF54 2AA400              LHLD  :00A4     Get input pointer
391 EF57 CDFBE6              CALL  :E6FB     Compare HL-DE
392 EF5A EB                  XCHG            Input pntr in E
393 EF5B 78                  MOV   A,B       Char in A
394 EF5C DC44EE              CC    :EE44     Space available: Find
395                                          current pos in buffer
396 EF5F D296EF              JNC   :EF96     Buffer full: quit, char in A
397 EF62 E5                  PUSH  H         Preserve end available space
398 EF63 2A7200              LHLD  :0072     Get cursor pos addr
399 EF66 E3                  XTHL            Put it on stack
400 EF67 E5                  PUSH  H
401 EF68 CD6BE3              CALL  :E36B     Delete cursor
402 EF6B EB                  XCHG            Old input pntr in HL
403 EF6C 00                  NOP
404 EF6D 23                  INX   H         +1
405 EF6E 22A400              SHLD  :00A4     Update input pointer
406 EF71 2B                  DCX   H
407 EF72 44                  MOV   B,H       ) Old input pntr in BC
408 EF73 4D                  MOV   C,L       )
409 EF74 2B                  DCX   H
410 EF75 EB                  XCHG            Old input pntr -1 in DE
411 EF76 E1                  POP   H         Get end available space
412 EF77 2B                  DCX   H         -1
413 EF78 CDC2E6              CALL  :E6C2     Move screen area 1 pos
414 EF7B 23                  INX   H
415 EF7C 77                  MOV   M,A       Insert char in buffer
416 EF7D 23                  INX   H
417 EF7E 22AE00              SHLD  :00AE     Preserve pntr to cursor
418                                          in buffer
419 EF81 E1                  POP   H         Get end available space
420 EF82 FE0D                CPI   :0D
421 EF84 CA9CEF              JZ    :EF9C     Jump if char is car.ret
422 EF87 CD29EF              CALL  :EF29     Reprint text line in window
423 EF8A CD30E3              CALL  :E330     Put cursor on screen
424 EF8D FE09                CPI   :09
425 EF8F CAB9EF              JZ    :EFB9     Jump if char is tab
426 EF92 CDF6ED              CALL  :EDF6     Move cursor right 1 pos
427 EF95 37          L2E218  STC
428 EF96 E1          L2E219  POP   H
429 EF97 D1                  POP   D
430 EF98 C1                  POP   B
431 EF99 78                  MOV   A,B
432 EF9A C1                  POP   B
433 EF9B C9                  RET
434
435                     * If car.ret:
```

```
436
437 EF9C AF          L2E220  XRA   A
438 EF9D 32A800                STA   :00A8     )
439 EFA0 32AB00                STA   :00AB     ) Reset pointers
440 EFA3 32AF00                STA   :00AF     )
441 EFA6 CD3AEF                CALL  :EF3A     Reprint full screen
442 EFA9 2AB000                LHLD  :00B0     Get pntr to start cursor
443                                            line on screen
444 EFAC 11F8FF                LXI   D,:FFF8
445 EFAF 19                    DAD   D         New cursor addr
446 EFB0 CD30E3                CALL  :E330     Put cursor on screen
447 EFB3 CDABED                CALL  :EDAB     Move cursor down
448 EFB6 C395EF                JMP   :EF95     Quit with CY=1
449
450                    * If tab:
451
452 EFB9 CDF6ED    L2E230  CALL  :EDF6     Move cursor right
453 EFBC 3AAB00                LDA   :00AB     Get X-offset of cursor
454                                            in document
455 EFBF 47                    MOV   B,A       in B
456 EFC0 2AB200                LHLD  :00B2     Get pntr to start cursor
457                                            line in buffer
458 EFC3 CD7BEE                CALL  :EE7B     Skip to Bth pos on line
459 EFC6 DAB9EF                JC    :EFB9     Again
460 EFC9 C395EF                JMP   :EF95     Abort with CY=1
461                    *
462                    ********************************
463                    * DELETE CHARACTER IN BUFFER *
464                    ********************************
465                    *
466                    * Deletes character at cursor position and moves
467                    * the text above this position. No action if past
468                    * car.ret, tab or at or beyond end of text.
469                    *
470                    * Exit: ABCDEHL preserved.
471                    *       CY=1: O.K.
472                    *       CY=0: If past car.ret, tab or beyond 0.
473                    *
474 EFCC C5          EDLCH   PUSH  B
475 EFCD F5                  PUSH  PSW
476 EFCE D5                  PUSH  D
477 EFCF E5                  PUSH  H
478 EFD0 CD44EE              CALL  :EE44     Find current pos in buffer
479 EFD3 D296EF              JNC   :EF96     Quit if past CR, tab
480                                            or 0 with CY=0
481 EFD6 7E                  MOV   A,M       Get char
482 EFD7 B7                  ORA   A
483 EFD8 CA96EF              JZ    :EF96     If at end of text:
484                                            quit with CY=0
485 EFDB 00                  NOP
486 EFDC 00                  NOP
487 EFDD 00                  NOP
488 EFDE E5                  PUSH  H
489 EFDF 2A7200              LHLD  :0072     Get cursor pos addr
490 EFE2 E3                  XTHL            preserve it on stack
491 EFE3 CD6BE3              CALL  :E36B     Delete cursor
492 EFE6 EB                  XCHG            Original HL in DE
493 EFE7 2AA400              LHLD  :00A4     Get input pointer
494 EFEA 2B                  DCX   H         Decrement it
495 EFEB 22A400              SHLD  :00A4     And store update input pntr
496 EFEE 44                  MOV   B,H       ) Move updated pointer
497 EFEF 4D                  MOV   C,L       ) into BC
```

```
498 EFF0 0B                 DCX   B         Decrement it once again
499 EFF1 EB                 XCHG            restore original HL
500 EFF2 CDC2E6             CALL  :E6C2     Move screen area above
501                                         downwards
502 EFF5 FE0D               CPI   :0D       Deleted car.ret ?
503 EFF7 C429EF             CNZ   :EF29     If not: reprint text line
504 EFFA CC3AEF             CZ    :EF3A     Else: reprint full screen
505 EFFD C3F2CE             JMP   :CEF2     Put cursor on screen, abort
506                                         with CY=1
507                 *
508                 *
509                 *
510 F000                    END
```

```
****************************
* S Y M B O L   T A B L E *
****************************
```

ECRT	EDF6	EDLCH	EFCC	EINCH	EF4B	L2E191	EE1C
L2E192	EE22	L2E193	EE33	L2E194	EE34	L2E195	EE38
L2E196	EE43	L2E197	EE44	L2E198	EE7B	L2E199	EE7F
L2E200	EE82	L2E201	EE94	L2E202	EE98	L2E203	EEA0
L2E204	EEA3	L2E205	EEA6	L2E206	EEAC	L2E207	EEBA
L2E208	EEC0	L2E209	EED2	L2E210	EEDE	L2E211	EEEB
L2E212	EEEC	L2E213	EF08	L2E214	EF17	L2E215	EF28
L2E216	EF29	L2E217	EF3A	L2E218	EF95	L2E219	EF96
L2E220	EF9C	L2E230	EFB9				

```
002                             ORG   :E000
003                     *
004                     *
005                     *
006                     *   ============================
007                     *** ENCODING / UTILITY PACKAGES ***
008                     *   ============================
009                     *
010                     * Called by RST 1; DATA XX. XX indicates the
011                     * offset of E000 for the different entrypoints.
012                     *
013                     * Contains also the key encoding routines and the
014                     * Heap organisation routines.
015                     *
016                     ****************
017                     * ENTRYPOINTS *
018                     ****************
019                     *
020 E000 C324E0         ELINE   JMP   :E024     Encode a BASIC line
021 E003 C32AE7         ELN     JMP   :E72A     Encode a linenr
022 E006 C345E1         ETCON   JMP   :E145     Encode a constant
023 E009 C374EA         USTART  JMP   :EA74     Start Utility package
024 E00C C390EF         DBOOT   JMP   :EF90     Disc bootstrap
025 E00F C39CE9         MHREQ   JMP   :E99C     Heap request
026 E012 C33FE9         MINKEY  JMP   :E93F     Encode a key input
027 E015 C335E9         L3E394  JMP   :E935     Get inputs from keyb
028                                               or DINC
029                     *
030                     *   ================
031                     *** ENCODING PACKAGE ***
032                     *   ================
033                     *
034                     * Generally in the encoding routines, HL points to
035                     * the first free location in the EBUF. C points to
036                     * the input text line.
037                     *
038                     ************************
039                     * UPDATE EBUF POINTER *
040                     ************************
041                     *
042                     * Increments the input pointer and checks if the
043                     * encoded input buffer (EBUF) is full.
044                     *
045                     * Entry: HL: input pointer EBUF.
046                     * Exit:  HL: updated pointer.
047                     *        Other registers preserved.
048                     *
049 E018 23              INXCH   INX   H         Incr pointer
050 E019 F5                     PUSH  PSW
051 E01A 7D                     MOV   A,L       Get lobyte in A
052 E01B FEBE                   CPI   :BE       Max value reached?
053 E01D 3E1A                   MVI   A,:1A     Buffer full ?
054 E01F D2F5D9                 JNC   :D9F5     Then run error 'LINE TOO
055                                               COMPLEX'
056 E022 F1                     POP   PSW
057 E023 C9                     RET
058                     *
059                     ****************************
060                     * ENCODE A BASIC COMMAND *
061                     ****************************
062                     *
                        * Looks for a match between input on line and the
```

```
064                    * BASIC command table #CBBF. Error exit 'COMMAND
065                    * INVALID' if the 2 high order bits of the code in
066                    * the table don't match the mask in D. Else the
067                    * code (#80 + low order 6 bits from code in table)
068                    * are stored in EBUF.
069                    *
070                    * Entry: HL: 1st free position in EBUF.
071                    *        D : Mask for direct command (#80) or
072                    *            program input (#40).
073                    *        C : Position on current line.
074                    * Exit:  HL: Updated.
075                    *        C : Points to 1st not used input in line.
076                    *        E : Code just stored.
077                    *        BD preserved, AF corrupted.
078                    *        Exit with jump to address found in table.
079                    *        #E04F is on stack as next returnaddress,
080                    *        entry DE is next on stack.
081                    *
082 E024 D5            L3E2    PUSH  D
083 E025 E5                    PUSH  H
084 E026 214FE0                LXI   H,:E04F   Returnaddr after finishing
085                                             encoding
086 E029 E3                    XTHL            Save it on stack
087 E02A E5                    PUSH  H         Save HL again
088 E02B 1E02                  MVI   E,:02     Addit. offset for finding
089                                             instruction in table
090 E02D 21BFCB                LXI   H,:CBBF   Startaddr table with strings
091                                             Basiccommands
092 E030 D5                    PUSH  D
093 E031 CD34CA                CALL  :CA34     Find instr in table
094 E034 D1                    POP   D         Get mask for direct/program
095 E035 7E                    MOV   A,M       Get instr code from table
096 E036 E6C0                  ANI   :C0       ) Check if it
097 E038 A2                    ANA   D         ) is a valid command
098 E039 3E18                  MVI   A,:18     If not: Run error
099 E03B CAF5D9                JZ    :D9F5     'COMMAND INVALID'
100 E03E 7E                    MOV   A,M       Get instr code from table
101 E03F 23                    INX   H
102 E040 E63F                  ANI   :3F       ) Make it a token
103 E042 F680                  ORI   :80       )
104 E044 5F                    MOV   E,A       Store token in E
105 E045 7E                    MOV   A,M       )
106 E046 23                    INX   H         ) Get addr of encoding
107 E047 66                    MOV   H,M       ) instruction in HL
108 E048 6F                    MOV   L,A       )
109 E049 E3                    XTHL            and save it on stack
110 E04A 73                    MOV   M,E       Store token in EBUF
111 E04B CD18E0                CALL  :E018     Update EBUF pointer
112 E04E C9                    RET             Go to encoding instruction
113                                             return afterwards to E04F
114                    *
115                    * End of encoding of an input line (after running
116                    * the command specific processing).
117                    * Checks if character after BASIC command is CR or
118                    * ':'.
119                    *
120                    * Entry: C : points to position on current line.
121                    *        On stack: Original DE.
122                    *
123 E04F D1            L3E3    POP   D         Get type of command in D
124 E050 CDD2DD                CALL  :DDD2     Get char from line, neglect
125                                             tab + space
```

```
126 E053 0C                 INR   C          Pntr to next char
127 E054 FE3A               CPI   :3A        ':' ?
128 E056 CA24E0             JZ    :E024      Then encode next instr
129 E059 FE0D               CPI   :0D        'CR' ?
130 E05B C20BDA             JNZ   :DA0B      Run 'SYNTAX ERROR' if not
131 E05E C9                 RET
132                         *
133                         ********************************
134                         * ENCODE 'FOR - TO - (STEP)' *
135                         ********************************
136                         *
137                         * Valid for all specific encoding routines:
138                         * HL points to 1st free EBUF location.
139                         *
140 E05F CDFEE0             EFOR  CALL  :E0FE      Encode 'LET'
141 E062 FE20               CPI   :20        String type ?
142 E064 CA1ADA             JZ    :DA1A      Then run error 'TYPE
143                                           MISMATCH'
144 E067 FE10               CPI   :10        INT type ?
145 E069 1188E0             LXI   D,:E088    Addr 'TO' table
146 E06C C39AE0             JMP   :E09A      Get addr encoding instr and
147                                           go to it
148
149                         * Encode 'TO':
150
151 E06F CC6DE3             L3E5  CZ    :E36D      Encode a INT expr
152 E072 C476E3                   CNZ   :E376      Encode a FPT expr
153 E075 1190E0                   LXI   D,:E090    Addr 'STEP' table
154 E078 C39AE0                   JMP   :E09A      Get addr encoding instr and
155                                                go to it
156
157                         * Encode 'STEP':
158
159 E07B CC6DE3             L3E6  CZ    :E36D      Encode a INT expr
160 E07E C476E3                   CNZ   :E376      Encode a FPT expr
161 E081 C9                       RET
162
163                         * End encoding:
164
165 E082 36FF               L3E363 MVI  M,:FF      FF in EBUF as separator
166 E084 CD18E0                    CALL :E018      Update EBUF pointer
167 E087 C9                        RET
168
169                         * Table:
170
171 E088 02                 L3E395 DATA :02
172 E089 54                        DATA :54        T
173 E08A 4F                        DATA :4F        O
174 E08B 6FE0                      DBL  :E06F      Addr encode 'TO'
175                         *
176 E08D 00                        DATA :00        'TO' not found:
177 E08E 0BDA                      DBL  :DA0B      Run 'SYNTAX ERROR'
178                         *
179 E090 04                 L3E396 DATA :04
180 E091 53                        DATA :53        S
181 E092 54                        DATA :54        T
182 E093 45                        DATA :45        E
183 E094 50                        DATA :50        P
184 E095 7BE0                      DBL  :E07B      Addr encode 'STEP'
185                         *
186 E097 00                        DATA :00        'STEP' not found:
187 E098 82E0                      DBL  :E082      End command
```

```
                        *
                        ************************************************
                        * GET ADDRESS ENCODING INSTRUCTION AND GO TO IT *
                        ************************************************
                        *
                        * Entry: DE : Points to table of format:
                        *             < length name / name / jumpaddr > or
                        *             < 00 / jumpaddr >.
                        *        C  : Points to input.
                        * Exit:  C  : Updated.
                        *        AFBHL preserved. D=0, E=1.
                        *
E09A E5         L3E7    PUSH  H
E09B F5                 PUSH  PSW
E09C EB                 XCHG            Addr table in HL
E09D 1E01               MVI   E,:01
E09F CD34CA             CALL  :CA34     Find instruction in table.
                                        On exit, HL points to addr
                                        of encoding routine
E0A2 7E                 MOV   A,M       )
E0A3 23                 INX   H         ) Get addr encoding instr
E0A4 66                 MOV   H,M       )  in HL
E0A5 6F                 MOV   L,A       )
E0A6 F1                 POP   PSW
E0A7 E3                 XTHL            Addr encoding instr on stack
E0A8 C9                 RET             Go to it
                        *
                        ****************************************
                        * ENCODE 'NEXT' AND 'NEXT <VARIABLE>' *
                        ****************************************
                        *
E0A9 CD59E8     ENEXT   CALL  :E859     Next char ':' or 'CR' ?
E0AC C8                 RZ              Then ready

                        * If NEXT <variable>:

E0AD 2B                 DCX   H
E0AE 34                 INR   M         Token +1 (#AC)
E0AF 23                 INX   H
E0B0 CDBCE5             CALL  :E5BC     Encode variable or array ref
E0B3 3A3601             LDA   :0136     Get type latest expression
E0B6 FE20               CPI   :20       String type ?
E0B8 CA1ADA             JZ    :DA1A     Then run error 'TYPE
                                        MISMATCH'
E0BB C9                 RET
                        *
                        ****************************************
                        * ENCODE 'IF - THEN' AND 'IF - GOTO' *
                        ****************************************
                        *
E0BC E5         EIF     PUSH  H
E0BD D5                 PUSH  D
E0BE CD9CE3             CALL  :E39C     Encode boolean expr (type
                                        #30)
E0C1 11EDE0             LXI   D,:E0ED   Addr 'THEN' table
E0C4 C39AE0             JMP   :E09A     Get addr encoding instr
                                        and go to it

                        * Encode 'THEN':

E0C7 CD31E7     L3E10   CALL  :E731     Read a linenr into EBUF
E0CA D1                 POP   D
```

```
250 E0CB D2D4E0              JNC   :E0D4     Jump if no linenr given
251
252                  * If linenr:
253
254 E0CE E3                   XTHL
255 E0CF 2B                   DCX   H
256 E0D0 34                   INR   M
257 E0D1 34                   INR   M        Token +2 (#A8)
258 E0D2 E1                   POP   H
259 E0D3 C9                   RET
260
261                  * If statement:
262
263 E0D4 F1           L3E11   POP   PSW
264 E0D5 E5                   PUSH  H        Preserve EBUF pntr
265 E0D6 CD18E0               CALL  :E018    Update EBUF pointer
266 E0D9 CD24E0               CALL  :E024    Encode BASIC command
267 E0DC 7D                   MOV   A,L      Lobyte new EBUF pntr in A
268 E0DD E3                   XTHL           Old EBUF pntr in HL
269 E0DE 95                   SUB   L
270 E0DF 3D                   DCR   A
271 E0E0 77                   MOV   M,A      Length string in EBUF entry
272 E0E1 E1                   POP   H
273 E0E2 0D                   DCR   C
274 E0E3 C9                   RET
275
276                  * Encode 'GOTO':
277
278 E0E4 F1           L3E12   POP   PSW
279 E0E5 E3                   XTHL
280 E0E6 2B                   DCX   H
281 E0E7 34                   INR   M        Token +1 (#A7)
282 E0E8 E1                   POP   H
283 E0E9 CD2AE7               CALL  :E72A    Get linenr
284 E0EC C9                   RET
285
286                  * Table:
287
288 E0ED 04          L3E397  DATA  :04
289 E0EE 54                  DATA  :54       T
290 E0EF 48                  DATA  :48       H
291 E0F0 45                  DATA  :45       E
292 E0F1 4E                  DATA  :4E       N
293 E0F2 C7E0                DBL   :E0C7     Addr encode 'THEN'
294                  *
295 E0F4 04                  DATA  :04
296 E0F5 47                  DATA  :47       G
297 E0F6 4F                  DATA  :4F       O
298 E0F7 54                  DATA  :54       T
299 E0F8 4F                  DATA  :4F       O
300 E0F9 E4E0                DBL   :E0E4     Addr encode 'GOTO'
301                  *
302 E0FB 00                  DATA  :00       If no 'THEN' or 'GOTO' found
303 E0FC 0BDA                DBL   :DA0B     Run 'SYNTAX ERROR'
304                  *
305                  ****************
306                  * ENCODE 'LET' *
307                  ****************
308                  *
309                  * Encodes a variable or array with arguments. Then
310                  * finds '=' in input (error if not). Then encodes
311                  * an expression, depending on variable type (error
```

```
                        * if type non-existing).
                        *
                        * Exit: DE: Offset of left-side variable.
                        *       A and B: Left-side type.
                        *       C, HL updated. F corrupted.
                        *
E0FE CDBCE5     ELET    CALL  :E5BC     Encode var or array ref
E101 CD67E8             CALL  :E867     Check next char is '='
E104 3D                 DATA  :3D
E105 3A3601             LDA   :0136     Get type latest expression
E108 FE00               CPI   :00       FPT ?
E10A CA76E3             JZ    :E376     Then encode FPT expr
E10D FE10               CPI   :10       INT ?
E10F CA6DE3             JZ    :E36D     Then encode INT expr
E112 C3A1E3     .       JMP   :E3A1     Else encode STR expr
                        *
                        ****************************************
                        * ENCODE 'INPUT' AND 'INPUT <STRING>' *
                        ****************************************
                        *
E115 CDD2DD     EINPUT  CALL  :DDD2     Get char from line, neglect
                                        tab + space
E118 FE22               CPI   :22       " ?
E11A C227E1             JNZ   :E127     Encode 'READ' if no string
                                        in INPUT statement

                        * If 'INPUT <string>':

E11D 2B                 DCX   H
E11E 34                 INR   M         Token +1 (#A1)
E11F 23                 INX   H
E120 CDA1E3             CALL  :E3A1     Encode STR expr
E123 CD67E8             CALL  :E867     Check if next char is ';'
E126 3B                 DATA  :3B
                                        Into EREAD
                        *
                        ******************
                        * ENCODE 'READ' *
                        ******************
                        *
                        * From L3E16 used by several routines, with another
                        * address in LXI D, ...., pointing to various kinds
                        * of encoding/get routines.
                        *
E127 11BCE5     EREAD   LXI   D,:E5BC   Addr routine encode variable
                                        or array reference
                        *
E12A E5         L3E16   PUSH  H
E12B 3600               MVI   M,:00     00 into EBUF (count)
E12D CD18E0             CALL  :E018     Update EBUF pointer
E130 D5         L3E17   PUSH  D
E131 CD64E1             CALL  :E164     Go to encoding routine

                        * Return here after encoding:

E134 D1                 POP   D
E135 E3                 XTHL
E136 34                 INR   M         Count in EBUF +1
E137 E3                 XTHL
E138 CDD2DD             CALL  :DDD2     Get char from line, neglect
                                        tab + space
E13B 0C                 INR   C         Points to next char on line
```

```
374 E13C FE2C               CPI   :2C       ',' ?
375 E13E CA30E1             JZ    :E130     Again if more items
376 E141 0D                 DCR   C         Correct line pointer
377 E142 33                 INX   SP
378 E143 33                 INX   SP        SP to returnaddr
379 E144 C9                 RET
380                 *
381                 ****************************************
382                 * ENCODE A NUMBER OR STRING CONSTANT *
383                 ****************************************
384                 *
385                 * Entry: A: Type.
386                 *
387 E145 F5         L3E18   PUSH  PSW       Preserve type
388 E146 B7                 ORA   A
389 E147 CA5EE1             JZ    :E15E     Jump if FPT type
390 E14A FE10               CPI   :10
391 E14C CA88E8             JZ    :E888     Jump if INT type
392
393                 * If STR type:
394
395 E14F CDD2DD             CALL  :DDD2     Get char from line, neglect
396                                         tab + space
397 E152 FE22               CPI   :22       '"' (quoted string) ?
398 E154 F5                 PUSH  PSW
399 E155 CC80E8             CZ    :E880     Then store quoted string
400                                         in EBUF
401 E158 F1                 POP   PSW
402 E159 C4A6E6             CNZ   :E6A6     Else store unquoted string
403                                         in EBUF
404 E15C F1         L3E19   POP   PSW
405 E15D C9                 RET
406
407                 * If FPT type:
408
409 E15E CD01E5      L3E20   CALL  :E501     Encode FPT nr into EBUF
410 E161 C393E8             JMP   :E893     Quit with evt 'SYNTAX ERROR'
411                 *
412                 ***************************
413                 * part of EREAD (3E12A) *
414                 ***************************
415                 *
416 E164 D5         L3E21   PUSH  D         Addr encoding routine on
417                                         stack
418 E165 C9                 RET             Go to it
419                 *
420                 ****************
421                 * ENCODE 'DIM' *
422                 ****************
423                 *
424 E166 116CE1     EDIM    LXI   D,:E16C   Addr routine 'encoding array
425                                         reference'
426 E169 C32AE1             JMP   :E12A     Continu encoding
427                 *
428                 ******************************
429                 * ENCODE AN ARRAY REFERENCE *
430                 ******************************
431                 *
432 E16C CDBCE5     L3E23   CALL  :E5BC     Encode var or array ref
433 E16F 78                 MOV   A,B       Type in A
434 E170 E640               ANI   :40       Array type ?
435 E172 CA29DA             JZ    :DA29     Run 'SUBSCRIPT ERROR' if not
```

```
436 E175 C9                 RET
437                 *
438                 ****************************************
439                 * ENCODE 'ON - GOTO' AND 'ON - GOSUB' *
440                 ****************************************
441                 *
442 E176 E5         EON     PUSH  H
443 E177 CD6DE3             CALL  :E36D     Encode INT expr
444 E17A 118DE1             LXI   D,:E18D   Addr table
445 E17D C39AE0             JMP   :E09A     Get addr encoding instr
446                                          and go to it
447
448                 * Encode linenr in GOSUB (L3E25) and GOTO (L3E411):
449
450 E180 E3         L3E25   XTHL
451 E181 2B                 DCX   H
452 E182 34                 INR   M         Token +1 (#AF)
453 E183 23                 INX   H
454 E184 E3                 XTHL
455 E185 E3         L3E411  XTHL
456 E186 E1                 POP   H
457 E187 112AE7             LXI   D,:E72A   Addr routine 'get linenr'
458 E18A C32AE1             JMP   :E12A     Continu encoding
459
460                 * Table:
461
462 E18D 04         L3E399  DATA  :04
463 E18E 47                 DATA  :47       G
464 E18F 4F                 DATA  :4F       O
465 E190 54                 DATA  :54       T
466 E191 4F                 DATA  :4F       O
467 E192 85E1               DBL   :E185     Addr encode 'GOTO'
468                 *
469 E194 05                 DATA  :05
470 E195 47                 DATA  :47       G
471 E196 4F                 DATA  :4F       O
472 E197 53                 DATA  :53       S
473 E198 55                 DATA  :55       U
474 E199 42                 DATA  :42       B
475 E19A 80E1               DBL   :E180     Addr encode 'GOSUB'
476                 *
477 E19C 00                 DATA  :00       If no GOTO or GOSUB found:
478 E19D 0BDA               DBL   :DA0B     Run 'SYNTAX ERROR'
479                 *
480                 ******************
481                 * ENCODE 'PRINT' *
482                 ******************
483                 *
484 E19F E5         EPRINT  PUSH  H
485 E1A0 3600               MVI   M,:00     00 into EBUF (init length)
486 E1A2 CD59E8             CALL  :E859     Next char ':' or 'CR' ?
487 E1A5 CACBE1             JZ    :E1CB     Then ready
488
489                 * If statement after PRINT:
490
491 E1A8 CD18E0     L3E27   CALL  :E018     Update EBUF pointer
492 E1AB E3                 XTHL
493 E1AC 34                 INR   M         Length +1
494 E1AD E3                 XTHL
495 E1AE CDB2E3             CALL  :E3B2     Encode non-boolean expr
496                                          preceeded by its type
497 E1B1 36FF               MVI   M,:FF     FF into EBUF
```

```
498 E1B3 CD59E8             CALL  :E859     Next char ':' or 'CR' ?
499 E1B6 CACBE1             JZ    :E1CB     Then ready
500
501                 * If more statements:
502
503 E1B9 77                 MOV   M,A       Char into EBUF
504 E1BA FE2C               CPI   :2C       ',' ?
505 E1BC CAC4E1             JZ    :E1C4     Then continu
506 E1BF FE3B               CPI   :3B       ';' ?
507 E1C1 C20BDA             JNZ   :DA0B     Run 'SYNTAX ERROR' if not
508 E1C4 0C         L3E28   INR   C         Update line pntr
509 E1C5 CD59E8             CALL  :E859     Next char ':' or 'CR' ?
510 E1C8 C2A8E1             JNZ   :E1A8     Continu if not
511                 *
512 E1CB E3         L3E29   XTHL
513 E1CC E1                 POP   H
514 E1CD CD18E0             CALL  :E018     Update EBUF pointer
515 E1D0 C9                 RET
516                 *
517                 ******************
518                 * ENCODE 'MODE' *
519                 ******************
520                 *
521 E1D1 16FF       EMODE   MVI   D,:FF     Default mode 0
522 E1D3 CDD2DD             CALL  :DDD2     Get char from line, neglect
523                                         tab + space
524 E1D6 0C                 INR   C         Points to next char
525 E1D7 D630               SUI   :30
526 E1D9 CAF1E1             JZ    :E1F1     If char is '0': FF in EBUF
527 E1DC DA0BDA             JC    :DA0B     Run 'SYNTAX ERROR' if not
528                                         number or printable char
529 E1DF FE07               CPI   :07       Between 0 and 7?
530 E1E1 D215DA             JNC   :DA15     Run error 'NUMBER OUT OF
531                                         RANGE'if not
532 E1E4 3D                 DCR   A         )
533 E1E5 87                 ADD   A         ) Calc code for mode
534 E1E6 57                 MOV   D,A       in D
535 E1E7 CDE0DD             CALL  :DDE0     Get next char from line
536 E1EA FE41               CPI   :41       'A' ?
537 E1EC C2F1E1             JNZ   :E1F1     Jump if not
538 E1EF 0C                 INR   C         Points to next char
539 E1F0 14                 INR   D         Set D for A-mode
540 E1F1 72         L3E31   MOV   M,D       Mode code in EBUF
541 E1F2 CD18E0             CALL  :E018     Update EBUF pointer
542 E1F5 C9                 RET
543                 *
544                 *
545                 *
546 E1F6                    END
```

```
****************************
* S Y M B O L   T A B L E *
****************************

DBOOT  E00C   EDIM   E166   EFOR   E05F   EIF    E0BC
EINPUT E115   ELET   E0FE   ELINE  E000   ELN    E003
EMODE  E1D1   ENEXT  E0A9   EON    E176   EPRINT E19F
EREAD  E127   ETCON  E006   INXCH  E018   L3E10  E0C7
L3E11  E0D4   L3E12  E0E4   L3E16  E12A   L3E17  E130
L3E18  E145   L3E19  E15C   L3E2   E024   L3E20  E15E
L3E21  E164   L3E23  E16C   L3E25  E180   L3E27  E1A8
L3E28  E1C4   L3E29  E1CB   L3E3   E04F   L3E31  E1F1
```

```
L3E363 E082   L3E394 E015   L3E395 E088   L3E396 E090
L3E397 E0ED   L3E399 E18D   L3E411 E185   L3E5   E06F
L3E6   E07B   L3E7   E09A   MHREQ  E00F   MINKEY E012
USTART E009
```

```
002                             ORG   :E1F6
003                     *
004                     *
005                     *
006                     **********************
007                     * ENCODE 'ENVELOPE' *
008                     **********************
009                     *
010                     * Encodes <ENV> (<V>,<T>;) <V>,<T> or
011                     * <ENV> (<V>,<T>;) <V>.
012                     *
013                     * Exit: HL points beyond expression in EBUF.
014                     *       AFBCDE corrupted.
015                     *
016 E1F6 CD6DE3         EENV    CALL  :E36D     Encode ENV nr
017 E1F9 E5                     PUSH  H         Preserve EBUF pntr
018 E1FA CD18E0                 CALL  :E018     Update EBUF pointer
019 E1FD 1600                   MVI   D,:00     Init length
020 E1FF CD22E2         L3E33   CALL  :E222     Encode <V>
021 E202 CD59E8                 CALL  :E859     Next char ':' or 'CR' ?
022 E205 CA1EE2                 JZ    :E21E     Then ready
023 E208 CD62E8                 CALL  :E862     Check if next char is ','
024                                               run error if not
025 E20B CD22E2                 CALL  :E222     Encode <T>
026 E20E CD67E8                 CALL  :E867     Check if next char is ';'
027 E211 3B                     DATA  :3B
028 E212 14                     INR   D         Length +1
029 E213 36FF                   MVI   M,:FF     FF into EBUF
030 E215 CD59E8                 CALL  :E859     Next char ':' or 'CR' ?
031 E218 C2FFE1                 JNZ   :E1FF     Again if not
032 E21B CD18E0                 CALL  :E018     Update EBUF pointer
033 E21E E3             L3E34   XTHL
034 E21F 72                     MOV   M,D       Length in EBUF after token
035 E220 E1                     POP   H
036 E221 C9                     RET
037                     *
038                     * ENCODE A <V> OR <T> ELEMENT:
039                     *
040                     * Exit: DE preserved.
041                     *
042 E222 D5             L3E35   PUSH  D
043 E223 CD6DE3                 CALL  :E36D     Encode INT expr
044 E226 D1                     POP   D
045 E227 C9                     RET
046                     *
047                     ****************************
048                     * ENCODE 'LIST' AND 'EDIT' *
049                     ****************************
050                     *
051                     * Checks the expression after the token and updates
052                     * the token on it.
053                     * On exit, the token is:
054                     *                       EDIT     LIST
055                     * Without linenr:        B6       93
056                     * One line:              B7       94
057                     * Part of program:       B8       95
058                     *
059                     * Entry: HL    : Points after token in EBUF.
060                     * Exit:  C, HL: Updated.
061                     *        D    : Token.
062                     *        AF corrupted, BE preserved.
063                     *
```

```
064                     ELIST
065 E228 2B             EEDIT   DCX   H          Pnts to token
066 E229 56                     MOV   D,M        Token in D
067 E22A E5                     PUSH  H          Preserve EBUF pntr
068 E22B 23                     INX   H
069 E22C CD59E8                 CALL  :E859      Next char ':' or 'CR' ?
070 E22F CA44E2                 JZ    :E244      Then ready
071 E232 CD48E2                 CALL  :E248      Read linenr into EBUF
072 E235 14                     INR   D          Token +1
073 E236 CD59E8                 CALL  :E859      Next char ':' or 'CR' ?
074 E239 CA44E2                 JZ    :E244      Then ready
075 E23C CD67E8                 CALL  :E867      Next char '-' ?
076 E23F 2D                     DATA  :2D
077 E240 CD48E2                 CALL  :E248      Read linenr into EBUF
078 E243 14                     INR   D          Again token +1
079 E244 E3             L3E37   XTHL
080 E245 72                     MOV   M,D        Token into EBUF
081 E246 E1                     POP   H
082 E247 C9                     RET
083                     *
084                     * READ LINENUMBER INTO EBUF:
085                     *
086                     * Reads a linenumber from the input line into
087                     * the EBUF. If no linenumber is given, 0000 is
088                     * inserted.
089                     *
090                     * Entry: HL: Points to 1st free location in EBUF.
091                     *         C : Points to input line.
092                     * Exit:   C, HL: Updated.
093                     *         AF corrupted, BDE preserved.
094                     *
095 E248 D5             L3E38   PUSH  D
096 E249 CD31E7                 CALL  :E731      Read linenr into EBUF
097 E24C D1                     POP   D
098 E24D D8                     RC               Ready if nr given
099 E24E 3600                   MVI   M,:00      00 into EBUF
100 E250 CD18E0                 CALL  :E018      Update EBUF pointer
101 E253 3600                   MVI   M,:00      00 into EBUF
102 E255 CD18E0                 CALL  :E018      Update EBUF pointer
103 E258 C9                     RET
104                     *
105                     ************************************************
106                     * ENCODE 'WAIT', 'WAIT TIME' AND 'WAIT MEM' *
107                     ************************************************
108                     *
109 E259 115FE2         EWAIT   LXI   D,:E25F    Addr table
110 E25C C39AE0                 JMP   :E09A      Get addr encoding instr
111                                              and go to it
112
113                     * Table:
114
115 E25F 04             L3E401  DATA  :04
116 E260 54                     DATA  :54        T
117 E261 49                     DATA  :49        I
118 E262 4D                     DATA  :4D        M
119 E263 45                     DATA  :45        E
120 E264 85E2                   DBL   :E285      Addr encode 'TIME'
121                     *
122 E266 03                     DATA  :03
123 E267 4D                     DATA  :4D        M
124 E268 45                     DATA  :45        E
125 E269 4D                     DATA  :4D        M
```

```
126 E26A 6FE2                   DBL   :E26F     Addr encode 'MEM'
127                     *
128 E26C 00                     DATA  :00
129 E26D 72E2                   DBL   :E272     Addr encode 'port'
130
131                     * Encode 'MEM':
132
133 E26F 2B             L3E40   DCX   H
134 E270 34                     INR   M         Token +1 (#91)
135 E271 23                     INX   H
136 E272 CD02E3         L3E412  CALL  :E302     Encode <INT expr>,<INT expr>
137 E275 36FF                   MVI   M,:FF     FF into EBUF
138 E277 CD18E0                 CALL  :E018     Update EBUF pointer
139 E27A CDD2DD                 CALL  :DDD2     Get char from line, neglect
140                                             tab + space
141 E27D FE2C                   CPI   :2C       ',' ?
142 E27F C0                     RNZ             Ready if not
143 E280 2B                     DCX   H
144 E281 03                     INX   B
145 E282 C36DE3                 JMP   :E36D     Encode INT expr
146
147                     * Encode 'TIME':
148
149 E285 2B             L3E41   DCX   H
150 E286 34                     INR   M
151 E287 34                     INR   M         Token +2 (#92)
152 E288 23                     INX   H
153 E289 C36DE3                 JMP   :E36D     Encode INT expr
154                     *
155                     ***********************************
156                     * ENCODE 'DRAW', 'FILL', 'DOT' *
157                     ***********************************
158                     *
159                     EFILL
160 E28C CD02E3         EDRAW   CALL  :E302     Encode <INT expr>,<INT expr>
161
162                     * Entry for encode 'DOT':
163
164 E28F CD02E3         EDOT    CALL  :E302     Idem
165 E292 C314E3                 JMP   :E314     Encode INT expr
166                     *
167                     ***************************************
168                     * ENCODE 'RUN' AND 'RUN <LINENR>' *
169                     ***************************************
170                     *
171 E295 CD59E8         ERUN    CALL  :E859     Next char ':' or 'CR' ?
172 E298 C8                     RZ              Then ready
173
174                     * If 'RUN <linenr>':
175
176 E299 2B                     DCX   H
177 E29A 34                     INR   M         Token +1 (#88)
178 E29B 23                     INX   H
179 E29C C32AE7                 JMP   :E72A     Get linenr into EBUF
180                     *
181                     ****************
182                     * ENCODE 'IMP' *
183                     ****************
184                     *
185                     * Note: This command is not encoded, but has
186                     * immediate effect.
```

```
188 E29F E5          EIMP    PUSH  H
189 E2A0 21F1E2              LXI   H,:E2F1   Startaddr table var.types
190 E2A3 1E00                MVI   E,:00     1 info byte
191 E2A5 CD34CA              CALL  :CA34     Find type in table
192 E2A8 7E                  MOV   A,M       Get IMP type from table
193 E2A9 B7                  ORA   A
194 E2AA FA0BDA              JM    :DA0B     'SYNTAX ERROR' if not found
195 E2AD F5                  PUSH  PSW       Preserve IMP type
196 E2AE FE20                CPI   :20       STR type ?
197 E2B0 CAB9E2              JZ    :E2B9     Then jump
198 E2B3 CD59E8              CALL  :E859     IMP INT/FPT alone ?
199 E2B6 CAD9E2              JZ    :E2D9     Then ready
200 E2B9 CDE6E2      EIM10   CALL  :E2E6     Get char from line; check if
201                                          upper case; INR C
202 E2BC F5                  PUSH  PSW       Save char in IMP instruction
203 E2BD CD67E8              CALL  :E867     Next char is '-' ?
204 E2C0 2D                  DATA  :2D
205 E2C1 CDE6E2              CALL  :E2E6     Get next char from line;
206                                          check if upper case; INR C
207 E2C4 213402              LXI   H,:0234   Base addr IMP table
208 E2C7 54                  MOV   D,H       ) into DE
209 E2C8 5D                  MOV   E,L       )
210 E2C9 CD30DE              CALL  :DE30     Calc offset end addr in HL
211 E2CC 23                  INX   H         +1
212 E2CD EB                  XCHG            In DE
213 E2CE F1                  POP   PSW       Get char
214 E2CF CD30DE              CALL  :DE30     Calc offset begin addr
215 E2D2 EB                  XCHG
216 E2D3 F1                  POP   PSW       Get IMP type
217 E2D4 CD7CDE      EIM20   CALL  :DE7C     Load given range with IMP
218                                          type
219 E2D7 E1                  POP   H         Re-instate 'encoded' pntr
220 E2D8 C9                  RET
221
222                  * If no range given:
223
224 E2D9 F1          L3E47   POP   PSW       Get IMP type
225 E2DA 328F02              STA   :028F     Store default number type
226 E2DD 117502              LXI   D,:0275   Startaddr impl. type table
227 E2E0 218F02              LXI   H,:028F   Addr after end table
228 E2E3 C3D4E2              JMP   :E2D4     Fill A-Z with reqd type
229
230                  * Get character from line and check if it is a
231                  * upper case character:
232
233 E2E6 CDD2DD      EALPHA  CALL  :DDD2     Get char from line, neglect
234                                          tab + space
235 E2E9 CD02DE              CALL  :DE02     Check if upper case char
236 E2EC D20BDA              JNC   :DA0B     Run 'SYNTAX ERROR' if not
237 E2EF 0C                  INR   C         Update textline pntr
238 E2F0 C9                  RET
239                  *
240                  * STRINGS VARIABLE TYPES TABLE:
241                  *
242                  * The first byte is a length byte. The byte
243                  * after the string is the variable type byte.
244                  *
245 E2F1 03          IMPTT   DATA  :03
246 E2F2 46                  DATA  :46       F
247 E2F3 50                  DATA  :50       P
248 E2F4 54                  DATA  :54       T
249 E2F5 00                  DATA  :00       type is #00
```

```
                        *
E2F6 03                         DATA  :03
E2F7 49                         DATA  :49       I
E2F8 4E                         DATA  :4E       N
E2F9 54                         DATA  :54       T
E2FA 10                         DATA  :10       type is #10
                        *
E2FB 03                         DATA  :03
E2FC 53                         DATA  :53       S
E2FD 54                         DATA  :54       T
E2FE 52                         DATA  :52       R
E2FF 20                         DATA  :20       type is #20
                        *
E300 00                         DATA  :00       End of table
E301 80                         DATA  :80
                        *
                        ************************************
                        * ENCODE 'POKE', 'OUT', 'CURSOR' *
                        ************************************
                        *
                        * Also used by: Encode 'DRAW' and 'FILL'.
                        *
                        EPOKE
                        EOUT
                        ECURS
E302 CD6DE3             ENC3    CALL  :E36D     Encode INT expr
E305 CD62E8                     CALL  :E862     Check if next char is ',';
                                                run error if not
E308 C36DE3                     JMP   :E36D     Encode INT expression
                        *
                        ******************************
                        * ENCODE 'COLORG', 'COLORT' *
                        ******************************
                        *
                        * Partly also used to encode 'CLEAR' and 'TALK'.
                        *
                        * Exit: C, HL: updated.
                        *       AFDE preserved, B corrupted.
                        *
                        ECOLT
E30B CD6DE3             ECOLG   CALL  :E36D     Encode INT expr
E30E CD6DE3                     CALL  :E36D     Idem
E311 CD6DE3             ENC6    CALL  :E36D     Idem

                        * Entry for encode CLEAR/TALK:

                        ETALK
E314 C36DE3             ECLEAR  JMP   :E36D     Idem
                        *
                        ****************************************
                        * ENCODE 'SOUND' WITH POSSIBLE 'OFF' *
                        ****************************************
                        *
                        * Encodes SOUND <CHAN><ENV><VOL><TG><FREQ>, or
                        * SOUND <CHAN> OFF or SOUND OFF.
                        *
                        * Exit: C, HL: Updated.
                        *       A preserved, B corrupted, D=0, E=1.
                        *       CY=1: Sound off.
                        *
E317 CD2CE3             ESOUND  CALL  :E32C     Encode a possible 'OFF'
E31A D8                         RC              Ready if 'OFF' given
```

```
312 E31B CD6DE3             CALL  :E36D     Encode <CHAN>
313 E31E CD25E3             CALL  :E325     Encode 'OFF' or <ENV><VOL>
314 E321 D8                 RC              Ready if 'OFF' given
315 E322 C311E3             JMP   :E311     Encode <TG><FREQ>
316                 *
317                 ****************************************
318                 * ENCODE 'NOISE' WITH POSSIBLE 'OFF' *
319                 ****************************************
320                 *
321                 * Encodes NOISE <ENV><VOL> or NOISE OFF.
322                 *
323                 * Exit: C, HL: Updated.
324                 *       CY=1 : Noise off.
325                 *       A preserved, B corrupted, D=0, E=1.
326                 *
327 E325 CD2CE3     ENOISE  CALL  :E32C     Encode  possible 'OFF'
328 E328 D411E3             CNC   :E311     Encode <ENV><VOL> if no
329                                         'OFF' given
330 E32B C9                 RET
331                 *
332                 ****************************
333                 * ENCODE A POSSIBLE 'OFF' *
334                 ****************************
335                 *
336                 * Exit: CY=1:  'OFF' in input; FF in EBUF.
337                 *       CY=0:  No 'OFF' given.
338                 *       C, HL: Updated.
339                 *       AB preserved, D=0, E=1.
340                 *
341 E32C 1132E3     L3E55   LXI   D,:E332   Startaddr table
342 E32F C39AE0             JMP   :E09A     Get addr encoding instr
343                                         and go to it
344
345                 * Table:
346
347 E332 03         L3E404  DATA  :03
348 E333 4F                 DATA  :4F       O
349 E334 46                 DATA  :46       F
350 E335 46                 DATA  :46       F
351 E336 3BE3               DBL   :E33B     addr encode 'OFF'
352                 *
353 E338 00                 DATA  :00
354 E339 42E3               DBL   :E342     encoding addr if no 'OFF'
355
356                 * Encode 'OFF':
357
358 E33B 36FF       L3E365  MVI   M,:FF     FF into EBUF
359 E33D CD18E0             CALL  :E018     Update EBUF pointer
360 E340 37                 STC             CY=1
361 E341 C9                 RET
362
363                 * If no 'OFF':
364
365 E342 B7         L3E366  ORA   A         CY=0
366 E343 C9                 RET
367                 *
368                 ******************
369                 * ENCODE 'CALLM' *
370                 ******************
371                 *
372 E344 CD6DE3     ECALM   CALL  :E36D     Encode memory addr (INT)
373 E347 36FF               MVI   M,:FF     FF into EBUF
```

```
374 E349 23                 INX   H
375 E34A CD59E8             CALL  :E859     Next char ':' or 'CR' ?
376 E34D C8                 RZ              Then ready
377 E34E 2B                 DCX   H
378 E34F CD62E8             CALL  :E862     Check if next char is ',';
379                                         run error if not
380 E352 C3BCE5             JMP   :E5BC     Encode var.pntr
381                 *
382                 ***************************
383                 * ENCODE 'SAVE', 'LOAD' *
384                 ***************************
385                 *
386                 * Checks if a name is given after 'SAVE/LOAD',
387                 * then encodes a string. Else #19,#00 (empty
388                 * unquoted string) is added to code.
389                 *
390                 * Exit: C,HL updated, AFB corrupted, DE preserved.
391                 *
392                 ELOAD
393 E355 CD59E8     ESAVE   CALL  :E859     Next char ':' or 'CR' ?
394 E358 C2A1E3             JNZ   :E3A1     Encode string if not
395 E35B 3619               MVI   M,:19     #19 in next loc EBUF
396 E35D CD18E0             CALL  :E018     Update EBUF pointer
397 E360 3600               MVI   M,:00     #00 in next loc EBUF
398 E362 CD18E0             CALL  :E018     Update EBUF pointer
399 E365 C9                 RET
400                 *
401                 ************************************
402                 * ENCODE 'REM' AND '***' (ERROR) *
403                 ************************************
404                 *
405                 EERR
406 E366 C3B0E6     EREM    JMP   :E6B0     Encode text
407                 *
408                 ***************************
409                 * ENCODE SINGLE ROUTINES *
410                 ***************************
411                 *
412                 EREST
413                 EEND
414                 ENEW
415                 ERET
416                 ECHECK
417                 ECONT
418                 ESTEP
419                 ETROF
420                 ESTOP
421                 ETRON
422 E369 C9         EUT     RET             No further handling
423                 *
424                 ***************************
425                 * ENCODE 'GOTO', 'GOSUB' *
426                 ***************************
427                 *
428                 EGOTO
429 E36A C32AE7     EGOSUB  JMP   :E72A     Get linenr
430                 *
431                 ************************************************
432                 * ENCODE AN EXPRESSION IF VARIABLE TYPE IS INT *
433                 ************************************************
434                 *
435 E36D D5         L3E371  PUSH  D
```

```
436 E36E 110001          LXI   D,:0100   Set D=#01 (conversion) and
437                                      E=#00 (FPT var type)
438 E371 0610            MVI   B,:10     Reqd type is INT
439 E373 C37CE3          JMP   :E37C     Encode expression
440                *
441                ************************************************
442                * ENCODE AN EXPRESSION IF VARIABLE TYPE IS FPT *
443                ************************************************
444                *
445 E376 D5        L3E372  PUSH  D
446 E377 111002          LXI   D,:0210   Set D for evt conversion
447                                      and E=#10 (INT var type)
448 E37A 0600            MVI   B,:00     Reqd type is FPT
449 E37C F5        L3E373  PUSH  PSW
450 E37D 78              MOV   A,B
451 E37E 329002          STA   :0290     Set reqd number type
452 E381 E5              PUSH  H
453 E382 D5              PUSH  D
454 E383 CDC9E3          CALL  :E3C9     Encode expr
455 E386 D1              POP   D
456 E387 3A3601          LDA   :0136     Get type latest expression
457 E38A B8              CMP   B         Was it as reqd ?
458 E38B CA98E3          JZ    :E398     Then ready
459
460                * Type not as expected:
461
462 E38E BB              CMP   E         Was it alternative type
463 E38F C21ADA          JNZ   :DA1A     Run error 'TYPE MISMATCH'
464                                      if not
465 E392 7A              MOV   A,D       Get conversion byte in A
466 E393 D1              POP   D
467 E394 D5              PUSH  D
468 E395 CD70E7          CALL  :E770     Add conversion byte to expr
469 E398 D1        L3E374  POP   D
470 E399 F1        L3E375  POP   PSW
471 E39A D1              POP   D
472 E39B C9              RET
473                *
474                ********************************
475                * ENCODE A BOOLEAN EXPRESSION *
476                ********************************
477                *
478                * Variable type is #30.
479                *
480 E39C 0630      L3E376  MVI   B,:30     Var type is #30
481 E39E C3A3E3          JMP   :E3A3     Encode expression
482                *
483                ********************************
484                * ENCODE A STRING EXPRESSION *
485                ********************************
486                *
487                * Variable type is #20.
488                *
489 E3A1 0620      L3E377  MVI   B,:20     Var type is #20
490
491                * Entry for 'encode Boolean expression':
492
493 E3A3 D5        L3E378  PUSH  D
494 E3A4 F5                PUSH  PSW
495 E3A5 CDC9E3            CALL  :E3C9     Encode expr
496 E3A8 3A3601            LDA   :0136     Get type latest expression
497 E3AB B8                CMP   B         Compare with reqd type
```

```
498 E3AC C21ADA             JNZ   :DA1A     Run error 'TYPE MISMATCH'
499                                          if not identical
500 E3AF C399E3             JMP   :E399     Quit
501                 *
502                 ************************************
503                 * ENCODE A NON-BOOLEAN EXPRESSION *
504                 ************************************
505                 *
506                 * Encodes an entire expression, preceeded by its
507                 * type, into the EBUF. 'TYPE MISMATCH' error occurs
508                 * if expression is boolean.
509                 *
510                 * Exit: C,HL updated; B preserved.
511                 *       A: Type:      Type in TYPE.
512                 *       D: Orig. B    OLDOP,RGTPT,HOPPT preserved.
513                 *       E: OLDOP.
514                 *
515 E3B2 E5         EEXPI   PUSH  H
516 E3B3 CD18E0             CALL  :E018     Update EBUF pointer
517 E3B6 AF                 XRA   A
518 E3B7 329002             STA   :0290     Req. number type is #00
519 E3BA CDC9E3             CALL  :E3C9     Encode expr
520 E3BD 3A3601             LDA   :0136     Get type latest expression
521 E3C0 FE30               CPI   :30       Boolean ?
522 E3C2 CA1ADA             JZ    :DA1A     Then run error 'TYPE
523                                          MISMATCH'
524 E3C5 E3                 XTHL            Get old EBUF pntr
525 E3C6 77                 MOV   M,A       Type into EBUF
526 E3C7 E1                 POP   H         New EBUF pntr
527 E3C8 C9                 RET
528                 *
529                 ************************
530                 * ENCODE AN EXPRESSION *
531                 ************************
532                 *
533                 * Routine encodes an entire expression until no
534                 * operator (or INOT) is found. The expression may
535                 * begin with an unitary operator (highest priority).
536                 *
537                 * Exit: C,HL updated, B preserved.
538                 *       A,E: OLDOP;   D: Entry B.
539                 *       OLDOP, RGTPT, HOPPT preserved.
540                 *       Type of expr in TYPE.
541                 *       RGTOP: #00 or #1E.
542                 *
543 E3C9 EB         L3E380  XCHG            Save HL in DE
544 E3CA 60                 MOV   H,B       Var type in H
545 E3CB 3A3801             LDA   :0138     Get old priority operator
546 E3CE 6F                 MOV   L,A       in L
547 E3CF E5                 PUSH  H         Save it on stack
548 E3D0 2A3901             LHLD  :0139     Get pntr place for operator
549 E3D3 E5                 PUSH  H         Save it on stack
550 E3D4 2A3B01             LHLD  :013B     Get pntr to RGT operand of
551                                          last operator
552 E3D7 E5                 PUSH  H         Save it on stack
553 E3D8 EB                 XCHG
554 E3D9 AF                 XRA   A
555 E3DA 323801             STA   :0138     Reset old prio operator
556 E3DD CDF1E3             CALL  :E3F1     Encode a term with possible
557                                          unitary operator
558 E3E0 EB                 XCHG
559 E3E1 E1                 POP   H
```

```
560 E3E2 223B01              SHLD  :013B     Restore RGTPT
561 E3E5 E1                  POP   H
562 E3E6 223901              SHLD  :0139     Restore HOPPT
563 E3E9 E1                  POP   H
564 E3EA 44                  MOV   B,H       Restore B
565 E3EB 7D                  MOV   A,L
566 E3EC 323801              STA   :0138     Restore OLDOP
567 E3EF EB                  XCHG
568 E3F0 C9                  RET
569                  *
570                  *
571                  *
572 E3F1                     END
```

****
* SYMBOL TABLE *
****

EALPHA	E2E6	ECALM	E344	ECHECK	E369	ECLEAR	E314
ECOLG	E30B	ECOLT	E30B	ECONT	E369	ECURS	E302
EDOT	E28F	EDRAW	E28C	EEDIT	E22B	EEND	E369
EENV	E1F6	EERR	E366	EEXPI	E3B2	EFILL	E28C
EGOSUB	E36A	EGOTO	E36A	EIM10	E2B9	EIM20	E2D4
EIMP	E29F	ELIST	E228	ELOAD	E355	ENC3	E302
ENC6	E311	ENEW	E369	ENOISE	E325	EOUT	E302
EPOKE	E302	EREM	E366	EREST	E369	ERET	E369
ERUN	E295	ESAVE	E355	ESOUND	E317	ESTEP	E369
ESTOP	E369	ETALK	E314	ETROF	E369	ETRON	E369
EUT	E369	EWAIT	E259	IMPTT	E2F1	L3E33	E1FF
L3E34	E21E	L3E35	E222	L3E365	E33B	L3E366	E342
L3E37	E244	L3E371	E36D	L3E372	E376	L3E373	E37C
L3E374	E398	L3E375	E399	L3E376	E39C	L3E377	E3A1
L3E378	E3A3	L3E38	E248	L3E380	E3C9	L3E40	E26F
L3E401	E25F	L3E404	E332	L3E41	E285	L3E412	E272
L3E47	E2D9	L3E55	E32C				

```
                        ORG   :E3F1
                *
                *
                *
                ******************************************************
                * ENCODE A TERM WITH A POSSIBLE UNITARY OPERATOR *
                ******************************************************
                *
                * L3E381: First operand may be preceeded by unitary
                *         operator +, - or INOT.
                *         Then code byte preceeding 1st operand is:
                *                          +    -  INOT
                *                INT:     BC   BD   BE
                *                FPT:     9C   9D    *
                *
                * L3E382: Encodes a sequence of higher priority
                *         operations and their operands.
                *         Encodes all terms after OLDOP in input
                *         which form a succession of higher priority
                *         operators until next lower or equal
                *         operator is found. This will be in RGTOP.
                *         The type of this collection will be in
                *         TYPE.
                *
                * Exit: C,HL updated. BE corrupted. CY=0.
                *       A: bits 5,6,7 OLDOP; D: bits 5,6,7 new RGTOP
                *
E3F1 CDF6E6     L3E381  CALL  :E6F6     Find unitary operator
                                        in table
E3F4 B7                 ORA   A
E3F5 C244E4             JNZ   :E444     Jump if found
                *
E3F8 223901     L3E382  SHLD  :0139     Set pntr place for operator
E3FB CD55E4             CALL  :E455     Encode 1st operand
E3FE CDEAE6     L3E383  CALL  :E6EA     Find binary or unitary
                                        operator in table
E401 323701             STA   :0137     Store latest prio operator
E404 3A3701     L3E384  LDA   :0137     Get latest prio operator
E407 E6E0               ANI   :E0       Prio in bits 5,6,7
E409 57                 MOV   D,A       RGTOP in D
E40A 3A3801             LDA   :0138     Get old prio operator
E40D E6E0               ANI   :E0       Prio only
E40F BA                 CMP   D         Compare both operators
E410 D0                 RNC             Ready if RGTOP <= OLDOP

                * RGTOP > OLDOP:

E411 EB                 XCHG
E412 2A3601             LHLD  :0136     Get type latest expression
E415 E5                 PUSH  H         Preserve type left operand
                                        and RGTOP
E416 3A3801             LDA   :0138     Get old prio operator
E419 F5                 PUSH  PSW       Preserve it
E41A D5                 PUSH  D         Preserve EBUF pntr
E41B 2A3901             LHLD  :0139     Get pntr place for operator
E41E E5                 PUSH  H         Preserve HOPPT
E41F EB                 XCHG            New EBUF pntr in HL
E420 3A3701             LDA   :0137     Get latest prio operator
E423 323801             STA   :0138     and store it as old one
E426 CDF8E3             CALL  :E3F8     Encode right hand operand
                                        until higher prio operators
E429 EB                 XCHG
```

```
064 E42A E1                 POP   H
065 E42B 223901             SHLD  :0139     Restore HOPPT
066 E42E E1                 POP   H
067 E42F 223B01             SHLD  :013B     Restore RGTPT
068 E432 F1                 POP   PSW
069 E433 323801             STA   :0138     Restore OLDOP
070 E436 EB                 XCHG            New EBUF pntr in HL
071 E437 D1                 POP   D         Restore type left operand
072                                         (E) and orig RGTOP (D)
073 E438 7A                 MOV   A,D       Old RGTOP in A
074 E439 E61F               ANI   :1F       Op.code only
075 E43B CDCFE7             CALL  :E7CF     Obtain type info for
076                                         binary operation
077 E43E CD57E7             CALL  :E757     Encode binary operation
078                                         into EBUF
079 E441 C304E4             JMP   :E404     Check again if prios correct
080
081                 * Unitary operator:
082
083 E444 223901      L3E56   SHLD  :0139     Set pntr place for operator
084 E447 E5                 PUSH  H
085 E448 CD55E4             CALL  :E455     Encode a term
086 E44B CD97E7             CALL  :E797     Encode unitary operator
087                                         for a term
088 E44E D1                 POP   D
089 E44F CD83E7             CALL  :E783     Byte in A into EBUF
090 E452 C3FEE3             JMP   :E3FE     Encode sequence with prio's
091                 *
092                 * ENCODE A TERM:
093                 *
094                 * Non-error exit: C,HL: updated.
095                 *                 BCDE corrupted, AF preserved.
096                 *
097 E455 F5          L3E57   PUSH  PSW
098 E456 CDD2DD             CALL  :DDD2     Get char from line, neglect
099                                         tab + space
100 E459 FE22               CPI   :22
101 E45B CA9BE4             JZ    :E49B     Jump if char is '"'
102 E45E FE28               CPI   :28
103 E460 CA86E4             JZ    :E486     Jump if char is '('
104 E463 CD02DE             CALL  :DE02     Check if char is upper case
105 E466 DA77E4             JC    :E477     Then jump
106 E469 FE2D               CPI   :2D
107 E46B CA0BDA             JZ    :DA0B     Run 'SYNTAX ERROR' if '-'
108 E46E CDC8E4             CALL  :E4C8     Encode a number
109 E471 D20BDA             JNC   :DA0B     Evt run 'SYNTAX ERROR'
110 E474 C3A4E4             JMP   .:E4A4    Store type and quit
111
112                 * If upper case character:
113
114 E477 CD22E5      L3E58   CALL  :E522     Encode function
115 E47A DAA4E4             JC    :E4A4     If ready: store type (#30)
116                                         and quit
117 E47D 48                 MOV   C,B
118 E47E 1600               MVI   D,:00
119 E480 CDBCE5             CALL  :E5BC     Encode var/array reference
120 E483 C3A7E4             JMP   :E4A7     Quit
121
122                 * If opening bracket:
123
124 E486 369A        L3E59   MVI   M,:9A     Load #9A in EBUF
125 E488 CD18E0             CALL  :E018     Update EBUF pointer
```

```
126 E48B 0C             INR   C         Next pos in textline
127 E48C CDC9E3         CALL  :E3C9     Encode expression
128 E48F CDE0DD         CALL  :DDE0     Get char from line
129 E492 FE29           CPI   :29
130 E494 C20BDA         JNZ   :DA0B     'SYNTAX ERROR' if not ')'
131 E497 0C             INR   C         Next pos in textline
132 E498 C3A7E4         JMP   :E4A7     Quit
133
134                 * If opening '"':
135
136 E49B 00         L3E60   NOP
137 E49C CD80E8             CALL  :E880     Store quoted text in EBUF
138 E49F 3E20               MVI   A,:20     Type is STR
139 E4A1 C3A4E4             JMP   :E4A4     Store type and quit
140
141                 * Ready:
142
143 E4A4 323601     L3E61   STA   :0136     Set type latest expression
144 E4A7 F1         L3E62   POP   PSW
145 E4A8 C9                 RET
146                 *
147                 ****************************
148                 * ENCODE 'SAVEA', 'LOADA' *
149                 ****************************
150                 *
151                 ELODA
152 E4A9 CD78E6     ESAVA   CALL  :E678     Enc. array without arguments
153 E4AC C355E3             JMP   :E355     Into encode 'SAVE/LOAD'
154                 *
155 E4AF FF                 DATA  :FF
156 E4B0 FF                 DATA  :FF
157 E4B1 FF                 DATA  :FF
158 E4B2 FF                 DATA  :FF
159 E4B3 FF                 DATA  :FF
160 E4B4 FF                 DATA  :FF
161 E4B5 FF                 DATA  :FF
162 E4B6 FF                 DATA  :FF
163 E4B7 FF                 DATA  :FF
164 E4B8 FF                 DATA  :FF
165 E4B9 FF                 DATA  :FF
166 E4BA FF                 DATA  :FF
167 E4BB FF                 DATA  :FF
168 E4BC FF                 DATA  :FF
169 E4BD FF                 DATA  :FF
170 E4BE FF                 DATA  :FF
171 E4BF FF                 DATA  :FF
172 E4C0 FF                 DATA  :FF
173 E4C1 FF                 DATA  :FF
174 E4C2 FF                 DATA  :FF
175 E4C3 FF                 DATA  :FF
176 E4C4 FF                 DATA  :FF
177 E4C5 FF                 DATA  :FF
178 E4C6 FF                 DATA  :FF
179 E4C7 FF                 DATA  :FF
180                 *
181                 ********************
182                 * ENCODE A NUMBER *
183                 ********************
184                 *
185                 * Encodes a INT or a FPT number.
186                 *
187                 * Entry: C : Offset of start of number.
```

```
                        *        HL: Points to EBUF.
                        *
                        * On exit: In A:     #00 FPT;  #10 INT;  #10 HEX.
                        *          In EBUF:  #10 FPT;  #14 INT;  #15 HEX.
                        *
                        * On exit, non-hex numbers will only be INT if reqd
                        * type or IMP type is INT, and there is no '.' or
                        * 'E' in the input string.
                        *
                        * Non-error exit: CY=1: C,HL updated, B preserved.
                        *                       A: Type-code, DE: entry BC.
                        * Error exit:     CY=0, BCDEHL preserved.
                        *
E4C8 C5         ENUM    PUSH  B
E4C9 E5                 PUSH  H
E4CA CDD2DD             CALL  :DDD2     Get char from line, neglect
                                        tab + space
E4CD FE23               CPI   :23
E4CF CA13E5             JZ    :E513     Hex if char is '#'
E4D2 3A9002             LDA   :0290     Get required number type
E4D5 B7                 ORA   A
E4D6 C2DCE4             JNZ   :E4DC     Jump if reqd type is not FPT
E4D9 3A8F02             LDA   :028F     Get default number type
E4DC FE00       ENM03   CPI   :00
E4DE CAFFE4             JZ    :E4FF     Jump if type is not FPT

                        * If INT:

E4E1 3614               MVI   M,:14     Load #14 in EBUF
E4E3 CD18E0             CALL  :E018     Update EBUF pointer
E4E6 CD7BE5             CALL  :E57B     INT nr into EBUF
E4E9 D2FFE4             JNC   :E4FF     Then handle as FPT
E4EC CDE0DD             CALL  :DDE0     Get char from line
E4EF FE2E               CPI   :2E
E4F1 CAFFE4             JZ    :E4FF     Try FPT if no digits before
                                        the '.'
E4F4 FE45               CPI   :45
E4F6 CAFFE4             JZ    :E4FF     Handle as FPT if 'E'
E4F9 3E10       ENM05   MVI   A,:10     Type is INT
E4FB D1         ENM10   POP   D
E4FC D1                 POP   D
E4FD 37                 STC             CY=1
E4FE C9                 RET

                        * If FPT:

E4FF E1         ENM20   POP   H
E500 C1                 POP   B

                        * Entry from ETCON:

E501 C5         ENM25   PUSH  B         ) Save pointers
E502 E5                 PUSH  H         )
E503 3610               MVI   M,:10     Load #10 in EBUF
E505 CD18E0             CALL  :E018     Update EBUF pointer
E508 CD96E5             CALL  :E596     FPT nr into EBUF
E50B 3E00               MVI   A,:00     Type is FPT
E50D DAFBE4             JC    :E4FB     Jump if no errors
E510 E1                 POP   H
E511 C1                 POP   B
E512 C9                 RET             Error exit (no number)
```

```
250                     * If HEX:
251
252 E513 3615           ENM30   MVI   M,:15      Load #15 in EBUF
253 E515 CD18E0                 CALL  :E018      Update EBUF pointer
254 E518 00                     NOP
255 E519 CD84E8                 CALL  :E884      Hex nr into EBUF
256 E51C D20BDA                 JNC   :DA0B      Run 'SYNTAX ERROR' if no
257                                              digits
258 E51F C3F9E4                 JMP   :E4F9      Quit
259                     *
260                     **********************
261                     * ENCODE A FUNCTION *
262                     **********************
263                     *
264                     * Reads a system function (both function and
265                     * arguments) into EBUF. Error exit if syntax or
266                     * type mismatch errors are found.
267                     *
268                     * Entry:   HL: Points to 1st free pos. in EBUF.
269                     *          C : Points to input.
270                     * Exit: If found: CY=1:
271                     *          A:  Type info of result.
272                     *          C,HL updated: BDE corrupted.
273                     *       If not found: CY=0:
274                     *          A:  End of table count.
275                     *          B:  Start of name in input.
276                     *          C:  Points beyond.
277                     *          DE: Points to 0 T/L byte at table end.
278                     *          HL: Points to EBUF.
279                     *
280 E522 E5             EFUN    PUSH  H
281 E523 37                     STC                Set 'include type letter'
282 E524 CDFDE6                 CALL  :E6FD        Find variable name in input,
283                                                 allow !,%,$.
284 E527 21E6CF                 LXI   H,:CFE6      Addr table BASIC functions
285 E52A CD5ACA                 CALL  :CA5A        Find function in table
286 E52D 7B                     MOV   A,E          Get serialnr of entry in
287                                                 table in A
288 E52E EB                     XCHG
289 E52F E1                     POP   H
290 E530 D27AE5                 JNC   :E57A        Abort if not found (CY=0)
291
292                     * If found in table:
293
294 E533 3620                   MVI   M,:20        Load fn.code (#20) in EBUF
295 E535 CD18E0                 CALL  :E018        Update EBUF pointer
296 E538 77                     MOV   M,A          Load how manyth function
297                                                 in EBUF
298 E539 CD18E0                 CALL  :E018        Update EBUF pointer
299 E53C 1A                     LDAX  D            Get T/L byte of function
300 E53D 13                     INX   D
301 E53E F5                     PUSH  PSW          Preserve it
302 E53F E60F                   ANI   :0F          Le ngth only
303 E541 CA76E5                 JZ    :E576        Jump if no arguments reqd
304
305                     * If arguments required:
306
307 E544 F5                     PUSH  PSW          Preserve length fuction
308 E545 CD67E8                 CALL  :E867        Check if next char is '(',
309 E548 28                     DATA  :28          run 'SYNTAX ERROR' if not
310 E549 EB             L3E72   XCHG
311 E54A 7E                     MOV   A,M          Get T/L byte
```

```
312 E54B 23                 INX   H
313 E54C EB                 XCHG
314 E54D FE30               CPI   :30       Boolean type ?
315 E54F CCBCE5             CZ    :E5BC     Then encode var/array ref.
316 E552 CA65E5             JZ    :E565     and jump
317 E555 FE20               CPI   :20       STR type ?
318 E557 CCA1E3             CZ    :E3A1     Then encode STR expr
319 E55A CA65E5             JZ    :E565     and jump
320 E55D FE10               CPI   :10       INT type ?
321 E55F CC6DE3             CZ    :E36D     Then encode INT expr
322 E562 C476E3             CNZ   :E376     Else: encode FPT expr
323                 *
324 E565 F1         L3E73   POP   PSW       Get length function
325 E566 3D                 DCR   A         Check if all arguments done
326 E567 CA72E5             JZ    :E572     Jump if ready
327 E56A F5                 PUSH  PSW       Preserve T/L function
328 E56B CD67E8             CALL  :E867     Check if next char is ','
329 E56E 2C                 DATA  :2C       run 'SYNTAX ERROR' if not
330 E56F C349E5             JMP   :E549     Encode next argument
331 E572 CD67E8     L3E74   CALL  :E867     Check if next char is ')',
332 E575 29                 DATA  :29       run 'SYNTAX ERROR' if not
333 E576 F1         L3E75   POP   PSW       Get T/L function
334 E577 E630               ANI   :30       Type only
335 E579 37                 STC             CY=1
336 E57A C9         L3E76   RET
337                 *
338                 ************************
339                 * INT NUMBER INTO EBUF *
340                 ************************
341                 *
342                 * Entry: C : Points to input.
343                 *         HL: Points to EBUF.
344                 *
345 E57B CDD2DD      EINT    CALL  :DDD2     Get char from line, neglect
346                                         tab + space
347 E57E FE2D                 CPI   :2D
348 E580 C285E5               JNZ   :E585     Jump if char is not '-'
349 E583 0C                 INR   C
350 E584 AF                 XRA   A         Else: clear sign bit
351 E585 CD24C0     L3E78   CALL  :C024     Input INT number to MACC
352 E588 C28DE5             JNZ   :E58D     Jump if nr was >= 0
353 E58B E7                 RST   4         Change sign MACC
354 E58C 60                 DATA  :60
355 E58D C3A3E5     L3E79   JMP   :E5A3     Move MACC into EBUF
356                 *
357                 ************************
358                 * HEX NUMBER INTO EBUF *
359                 ************************
360                 *
361 E590 CD2AC0     L3E80   CALL  :C02A     Input HEX number to MACC
362 E593 C3A3E5             JMP   :E5A3     Move MACC into EBUF
363                 *
364                 ************************
365                 * FPT NUMBER INTO EBUF *
366                 ************************
367                 *
368                 * Entry: C : Points to input.
369                 *         HL: Points to EBUF.
370                 * Non-error exit:  CY=1:
371                 *         C,HL updated. B preserved. A corrupted.
372                 *         DE: Location of number in EBUF.
373                 * Error exit: CY=0:
```

```
374                     *       BDEHL preserved, A corrupted, C updated.
375                     *
376 E596 CDD2DD         EFPT    CALL    :DDD2       Get char from line, neglect
377                                                 tab + space
378 E599 FE2D                   CPI     :2D
379 E59B C2A0E5                 JNZ     :E5A0       Jump if char is not '-'
380 E59E 0C                     INR     C
381 E59F AF                     XRA     A           Else: clear sign bit
382 E5A0 CD79E8         L3E82   CALL    :E879       FPT nr into MACC, evt
383                                                 change sign
384
385                     * Entry for HEX/INT numbers:
386
387 E5A3 D2BBE5         L3E83   JNC     :E5BB       Abort if there are no digits
388 E5A6 C5                     PUSH    B
389 E5A7 54                     MOV     D,H         Old EBUF pntr in DE
390 E5A8 5D                     MOV     E,L
391 E5A9 CD18E0                 CALL    :E018       ) Update EBUF pointer
392 E5AC CD18E0                 CALL    :E018       ) to after new input
393 E5AF CD18E0                 CALL    :E018       )
394 E5B2 CD18E0                 CALL    :E018       )
395 E5B5 EB                     XCHG
396 E5B6 E7                     RST     4           Copy MACC into EBUF
397 E5B7 0F                     DATA    :0F
398 E5B8 EB                     XCHG
399 E5B9 C1                     POP     B
400 E5BA 37                     STC                 CY=1
401 E5BB C9             L3E84   RET
402                     *
403                     *
404                     *
405 E5BC                        END
```

```
*****************************
* S Y M B O L   T A B L E *
*****************************
```

EFPT	E596	EFUN	E522	EINT	E57B	ELODA	E4A9
ENM03	E4DC	ENM05	E4F9	ENM10	E4FB	ENM20	E4FF
ENM25	E501	ENM30	E513	ENUM	E4C8	ESAVA	E4A9
L3E381	E3F1	L3E382	E3F8	L3E383	E3FE	L3E384	E404
L3E56	E444	L3E57	E455	L3E58	E477	L3E59	E486
L3E60	E49B	L3E61	E4A4	L3E62	E4A7	L3E72	E549
L3E73	E565	L3E74	E572	L3E75	E576	L3E76	E57A
L3E78	E585	L3E79	E58D	L3E80	E590	L3E82	E5A0
L3E83	E5A3	L3E84	E5BB				

```
002                             ORG   :E5BC
003                     *
004                     *
005                     *
006                     ****************************************
007                     * ENCODE VARIABLE OR ARRAY REFERENCE *
008                     ****************************************
009                     *
010                     * L3E85: Reference to a variable or an array with
011                     *        arguments.
012                     * L3E86: Reference to an array without arguments.
013                     *
014                     * Entry: D : Code: 00: Reference to a value (array
015                     *                      with arguments or variable).
016                     *                  FF: Array name without arguments.
017                     *        C : Next position in input.
018                     *        HL: 1st free position in EBUF.
019                     * Exit:  C,HL updated, AF preserved.
020                     *        DE: Offset to symbol table (to T/L byte).
021                     *        B : T/L byte of name.
022                     *
023 E5BC 1600           L3E85   MVI   D,:00
024 E5BE F5             EVR10   PUSH  PSW
025 E5BF D5                     PUSH  D
026 E5C0 CDD2DD                 CALL  :DDD2     Get 1st char from line,
027                                             neglect tab + space
028 E5C3 CD02DE                 CALL  :DE02     Check if char is upper case
029 E5C6 D20BDA                 JNC   :DA0B     Run 'SYNTAX ERROR' if not
030 E5C9 E5                     PUSH  H
031 E5CA F5                     PUSH  PSW       Save 1st char
032
033                     * Check if name is a BASIC command:
034
035 E5CB 1E02                   MVI   E,:02     Nr of info bytes -1
036 E5CD 21BFCB                 LXI   H,:CBBF   Addr command table
037 E5D0 CD34CA                 CALL  :CA34     Find instr in table. On
038                                             exit, HL points to string
039                                             or after it if not found
040 E5D3 7E                     MOV   A,M       )
041 E5D4 E63F                   ANI   :3F       ) Check if end table reached
042 E5D6 FE25                   CPI   :25       )
043 E5D8 C20BDA                 JNZ   :DA0B     Run 'SYNTAX ERROR' if name
044                                             is a command
045 E5DB B7                     ORA   A
046
047                     * Check if name is a BASIC function:
048
049 E5DC CDFDE6                 CALL  :E6FD     Find var.name in input
050 E5DF 21E6CF                 LXI   H,:CFE6   Addr function table
051 E5E2 CD5ACA                 CALL  :CA5A     Find function in table
052 E5E5 DA0BDA                 JC    :DA0B     Run 'SYNTAX ERROR' if name
053                                             is a function
054
055                     * Check type marker in input:
056
057 E5E8 0C                     INR   C         Points to next char in input
058 E5E9 1E02                   MVI   E,:02     2 bytes in symtab for STR
059 E5EB 2620                   MVI   H,:20     String type byte
060 E5ED FE24                   CPI   :24
061 E5EF CA0EE6                 JZ    :E60E     Jump if STR ('$')
062 E5F2 1E04                   MVI   E,:04     4 byte in symtab for INT/FPT
063 E5F4 2610                   MVI   H,:10     INT type byte
```

```
064 E5F6 FE25               CPI   :25
065 E5F8 CA0EE6             JZ    :E60E     Jump if INT ('%')
066 E5FB 2600               MVI   H,:00     FPT type byte
067 E5FD FE21               CPI   :21
068 E5FF CA0EE6             JZ    :E60E     Jump if FPT ('!')
069
070                 * If no type marker given:
071
072 E602 F1                 POP   PSW       Get 1st byte var.name
073 E603 213402             LXI   H,:0234   Baseaddr IMPTAB
074 E606 CD30DE             CALL  :DE30     Calc offset addr in HL
075 E609 66                 MOV   H,M       Get type marker in H
076 E60A 0D                 DCR   C
077 E60B C30FE6             JMP   :E60F
078
079                 * Handle type marker:
080
081 E60E F1         L3E87   POP   PSW       Get 1st byte of name
082 E60F 7C         L3E88   MOV   A,H       type in A
083 E610 B2                 ORA   D         OR type (high nibble) with
084                                         length (low nibble)
085 E611 57                 MOV   D,A       T/L on name in D
086 E612 E1                 POP   H
087 E613 F1                 POP   PSW       Get code 00/FF in A
088 E614 E5                 PUSH  H
089 E615 CD18E0             CALL  :E018     ) Update EBUF pointer
090 E618 CD18E0             CALL  :E018     ) 2 positions
091 E61B B7                 ORA   A         Flags on code
092 E61C CA26E6             JZ    :E626     Jump if value
093
094                 * If name:
095
096 E61F 7A                 MOV   A,D       Get T/L byte of name
097 E620 F640               ORI   :40       Set bit 6 (array)
098 E622 57                 MOV   D,A       Preserve it
099 E623 C32EE6             JMP   :E62E
100
101                 * If value:
102
103 E626 CDE0DD     L3E89   CALL  :DDE0     Get char from line
104 E629 FE28               CPI   :28       '(' ?
105 E62B CC53E6             CZ    :E653     Then encode arguments
106                 *
107 E62E E3         L3E90   XTHL
108 E62F E5                 PUSH  H
109 E630 7A                 MOV   A,D       Get T/L byte on name
110 E631 E630               ANI   :30       Type only
111 E633 323601             STA   :0136     Set type latest expression
112 E636 D5                 PUSH  D         Preserve T/L name
113 E637 CD57CA             CALL  :CA57     Find variable in symtab
114 E63A D1                 POP   D         Get T/L name
115 E63B D47DE6             CNC   :E67D     Insert variable in symtab
116                                         if it is a new one
117 E63E 42                 MOV   B,D       T/L name in B
118 E63F EB                 XCHG            Var.addr in symtab in DE
119 E640 2AA102             LHLD  :02A1     Get startaddr symtab
120 E643 EB                 XCHG            in DE; var.addr in HL
121 E644 CD1ADE             CALL  :DE1A     Calc offset from begin
122                                         symtab in HL
123 E647 EB                 XCHG            Offset in DE
124 E648 E1                 POP   H         Retrieve EBUF pntr
125 E649 7A                 MOV   A,D       Hibyte offset in A
```

```
126 E64A F640               ORI   :40       Set bit 6 (array)
127 E64C 77                 MOV   M,A       Hibyte offset in EBUF
128 E64D 23                 INX   H
129 E64E 73                 MOV   M,E       Lobyte offset in EBUF
130 E64F 23                 INX   H
131 E650 E1                 POP   H
132 E651 F1                 POP   PSW
133 E652 C9                 RET
134                 *
135                 * ENCODE ARRAY ARGUMENTS:
136                 *
137                 * An arguments list is encoded into EBUF.
138                 * Format:
139                 *       nr of arg / type of arg / code for expr/
140                 *       < type of arg / code for expr >.
141                 *
142                 * Entry: D : T/L byte of variable name.
143                 *        C : Points to '(' of argument list for
144                 *            array in input.
145                 *        HL: 1st free position EBUF.
146                 * Exit:  D : 'Subscripted' flag.
147                 *        E : Nr of bytes in symtab (02).
148                 *        C,HL updated. B preserved. A=D.
149                 *
150 E653 1E00       EVR50  MVI   E,:00      Parameter count
151 E655 E5                  PUSH  H
152 E656 CD18E0            CALL  :E018     Update EBUF pointer
153 E659 1C       EVR55  INR   E          Parameter count +1
154 E65A 0C                INR   C         Skip 'C' or ','
155 E65B D5                PUSH  D
156 E65C CDB2E3            CALL  :E3B2     Encode non-boolean expr
157                                        preceeded by its type
158 E65F D1                POP   D
159 E660 CDD2DD            CALL  :DDD2     Get char from line, neglect
160                                        tab + space
161 E663 FE2C              CPI   :2C
162 E665 CA59E6            JZ    :E659     Get next parameter if it
163                                        is ','
164 E668 FE29              CPI   :29
165 E66A C20BDA            JNZ   :DA0B     Run 'SYNTAX ERROR' if
166                                        not ')'
167 E66D 0C                INR   C         Skip ')'
168 E66E E3                XTHL            Get old EBUF pntr
169 E66F 73                MOV   M,E       Parameter count into EBUF
170 E670 E1                POP   H
171 E671 1E02              MVI   E,:02     2 bytes space in symtab
172 E673 7A                MOV   A,D
173 E674 F640              ORI   :40       Set type is array
174 E676 57                MOV   D,A       Set flag 'subscripted'
175 E677 C9                RET
176                 *
177                 ************************
178                 * ENCODE AN ARRAY NAME *
179                 ************************
180                 *
181 E678 16FF       EARRN  MVI   D,:FF     Code for name only
182 E67A C3BEE5            JMP   :E5BE     Encode array name
183                 *
184                 ****************************************
185                 * INSERT A NEW VARIABLE IN SYMBOL TABLE *
186                 ****************************************
187                 *
```

```
188                     * The variable name is inserted in the symbol table
189                     * and the value is cleared.
190                     *
191                     * Entry: See CAB8.
192                     * Exit:  HL: Points to 2nd T/L byte of entry.
193                     *         AF corrupted, BCDE preserved.
194                     *
195 E67D CDB8CA         EVARI   CALL  :CAB8     Insert var.name in symtab
196 E680 E5                     PUSH  H
197 E681 23                     INX   H         HL pnts after 2nd T/L byte
198                                             of entry
199 E682 7A                     MOV   A,D       Get T/L of name
200 E683 E640                   ANI   :40
201 E685 C295E6                 JNZ   :E695     Jump if array type
202
203                     * If number type:
204
205 E688 7A                     MOV   A,D       Get T/L of name
206 E689 E630                   ANI   :30
207 E68B FE20                   CPI   :20
208 E68D CA95E6                 JZ    :E695     Jump if string type
209 E690 CD9ECB                 CALL  :CB9E     Clear value in symtab
210 E693 E1                     POP   H
211 E694 C9                     RET
212
213                     * If string/array type:
214
215 E695 3600           EVI10   MVI   M,:00     ) Clear pointer in symbtab
216 E697 23                     INX   H         )
217 E698 3600                   MVI   M,:00     )
218 E69A E1                     POP   H
219 E69B C9                     RET
220                     *
221                     ******************************
222                     * STORE QUOTED TEXT IN EBUF *
223                     ******************************
224                     *
225 E69C 3618           L3E96   MVI   M,:18     Code for quoted string (#18)
226                                             into EBUF
227 E69E CD18E0                 CALL  :E018     Update EBUF pointer
228 E6A1 1EFF                   MVI   E,:FF     Text must end with '"'
229 E6A3 C3B5E6                 JMP   :E6B5     Into common end
230                     *
231                     ************************************
232                     * STORE UNQUOTED STRING IN EBUF *
233                     ************************************
234                     *
235 E6A6 1E01           L3E97   MVI   E,:01     Text must end with ','
236 E6A8 3619                   MVI   M,:19     Code for unquoted string
237                                             (#19) into EBUF
238 E6AA CD18E0                 CALL  :E018     Update EBUF pointer
239 E6AD C3B5E6                 JMP   :E6B5     Into common end
240                     *
241                     ************************
242                     * STORE TEXT INTO EBUF *
243                     ************************
244                     *
245                     * Text in DATA, REM and '***' statements is moved
246                     * into the EBUF.
247                     *
248 E6B0 CDD2DD         L3E98   CALL  :DDD2     Get char from line, neglect
249                                             tab + space
```

```
250 E6B3 1E02                MVI   E,:02     Text must end with CR
251                                           Into common end
252                  *
253                  ************************************
254                  * COMMON END TEXT ENCODING ROUTINES *
255                  ************************************
256                  *
257                  * Entry: C : Points to 1st actual character to be
258                  *            stored.
259                  *        HL: Points to place for length byte in EBUF
260                  *        E : Handling switch:
261                  *             > 1: (but <#80): Text must end with CR
262                  *             = 1: Text will end with ',' (',' is no
263                  *                  inserted into EBUF).
264                  *            <= 0: Text will end at '"' ('"' is not
265                  *                  inserted into the EBUF).
266                  * Exit:  C : Points beyond text in input.
267                  *        HL: Points beyond stored text in EBUF.
268                  *        D : Length of stored text.
269                  *        A : Character which marks end of text.
270                  *        B preserved, E corrupted.
271                  *
272 E6B5 E5        L3E99   PUSH  H
273 E6B6 CD18E0            CALL  :E018     Update EBUF pointer
274 E6B9 1600            MVI   D,:00     Set length is 0
275 E6BB CDE0DD    L3E100  CALL  :DDE0     Get char from line
276 E6BE FE0D            CPI   :0D
277 E6C0 CADBE6          JZ    :E6DB     Jump if char is 'CR'
278 E6C3 FE2C            CPI   :2C
279 E6C5 CAE3E6          JZ    :E6E3     Jump if char is ','
280 E6C8 0C        L3E101  INR   C
281 E6C9 FE22            CPI   :22
282 E6CB C2D3E6          JNZ   :E6D3     Jump if char is not '"'
283 E6CE 1D              DCR   E
284 E6CF FADFE6          JM    :E6DF     If done: store length in
285                                       EBUF, quit
286 E6D2 1C              INR   E
287
288                  * Character into EBUF:
289
290 E6D3 77        L3E102  MOV   M,A       Load char in EBUF
291 E6D4 CD18E0            CALL  :E018     Update EBUF pointer
292 E6D7 14              INR   D
293 E6D8 C3BBE6          JMP   :E6BB     Get next char
294
295                  * If 'CR':
296
297 E6DB 1D        L3E103  DCR   E
298 E6DC FA0BDA          JM    :DA0B     If E >= #80: Run 'SYNTAX
299                                       ERROR'
300 E6DF E3        L3E104  XTHL
301 E6E0 72              MOV   M,D       Length in EBUF entry
302 E6E1 E1              POP   H
303 E6E2 C9              RET
304
305                  * If ',':
306
307 E6E3 1D        L3E105  DCR   E
308 E6E4 CADFE6          JZ    :E6DF     If E=0: Store length in
309                                       EBUF, quit
310 E6E7 C3ABE8          JMP   :E8AB     incr E, get next char
311                  *
```

```
                        ***********************************************
                        * FIND BINARY OR UNITARY OPERATOR IN TABLE *
                        ***********************************************
                        *
                        * Entry/exit: See #3E6F6.
                        *
E6EA E5         L3E106  PUSH  H
E6EB 2191CF             LXI   H,:CF91   Startaddr table
E6EE 1E00       L3E107  MVI   E,:00
E6F0 CD34CA             CALL  :CA34     Find instr in table
E6F3 7E                 MOV   A,M       Get code from table
E6F4 E1                 POP   H
E6F5 C9                 RET
                        *
                        ***************************************
                        * FIND AN UNITARY OPERATOR IN TABLE *
                        ***************************************
                        *
                        * Routine looks for a init. string beginning at
                        * C in table.
                        *
                        * Entry: C : Points to input.
                        * Exit:  CY=0: Not found:
                        *              C : Points to 1st valid character
                        *                  after entry address.
                        *              A : Contains code info 0.
                        *              DE = 0, BHL preserved.
                        *        CY=1: Found:
                        *              C : Points beyond string found.
                        *              A : Code byte from table.
                        *              DE = 0, BHL preserved.
                        *
E6F6 E5         L3E108  PUSH  H
E6F7 21D8CF             LXI   H,:CFD8   Startaddr table
E6FA C3EEE6             JMP   :E6EE     Into previous routine
                        *
                        *
                        *
E6FD                    END
```

```
*****************************
* S Y M B O L   T A B L E *
*****************************

EARRN  E678   EVARI  E67D   EVI10  E695   EVR10  E5BE
EVR50  E653   EVR55  E659   L3E100 E6BB   L3E101 E6C8
L3E102 E6D3   L3E103 E6DB   L3E104 E6DF   L3E105 E6E3
L3E106 E6EA   L3E107 E6EE   L3E108 E6F6   L3E85  E5BC
L3E87  E60E   L3E88  E60F   L3E89  E626   L3E90  E62E
L3E96  E69C   L3E97  E6A6   L3E98  E6B0   L3E99  E6B5
```

```
002                             ORG   :E6FD
003                     *
004                     *
005                     *
006                     ********************************
007                     * FIND VARIABLE NAME IN INPUT *
008                     ********************************
009                     *
010                     * Checks if 1st character is a upper case one.
011                     * Reads the input (starting with character after
012                     * C) till it finds a non-alphanumeric character
013                     * (number or upper case). On a carry CALL, it also
014                     * accepts %, ! or $ at the end. Blanks are not
015                     * ignored and not accepted.
016                     *
017                     * Entry: C :  Input position.
018                     * Exit:  B :  Entry C.
019                     *        C :  Points to 1st character not accepted.
020                     *        D :  Count of 1st character not accepted.
021                     *             (1st character read has count 1).
022                     *        A,E: 1st non-alphanumeric character read.
023                     *        HL preserved, F corrupted.
024                     *
025 E6FD F5             RDID    PUSH  PSW       Preserve CY-flag
026 E6FE 41                     MOV   B,C
027 E6FF 1600                   MVI   D,:00     Init count
028 E701 14             L3E110  INR   D         Count +1
029 E702 0C                     INR   C         Line pos +1
030 E703 7A                     MOV   A,D       Count in A
031 E704 FE0F                   CPI   :0F       Max 14 char for a name
032 E706 DA0AE7                 JC    :E70A
033 E709 15                     DCR   D         Skip last char if > 14
034 E70A CDE0DD         L3E111  CALL  :DDE0     Get char from line
035 E70D CD09DE                 CALL  :DE09     Check if nr or upper case
036 E710 DA01E7                 JC    :E701     Get next char if O.K.
037 E713 CDE0DD                 CALL  :DDE0     Get 1st non-alphanum.char
038                                             from line
039 E716 5F                     MOV   E,A       Store it in E
040 E717 F1                     POP   PSW       Get CY-flag back
041 E718 7B                     MOV   A,E       Get char back in A
042 E719 D0                     RNC             Abort if non-carry CALL
043
044                     * On carry CALL only: accept !,%,$:
045
046 E71A FE25                   CPI   :25
047 E71C CA27E7                 JZ    :E727     Jump if char is '%'
048 E71F FE21                   CPI   :21
049 E721 CA27E7                 JZ    :E727     Jump if char is '!'
050 E724 FE24                   CPI   :24
051 E726 C0                     RNZ             Abort if char is not '$'
052 E727 0C             L3E112  INR   C         Update line pos
053 E728 14                     INR   D         Update count
054 E729 C9                     RET
055                     *
056                     ********************
057                     * GET LINE NUMBER *
058                     ********************
059                     *
060                     * Exit: C,HL: Updated.
061                     *       B preserved, AFDE corrupted.
062                     *       'SYNTAX ERROR' if no linenr given.
```

```
064 E72A CD31E7     ELN     CALL  :E731     Read linenr into EBUF
065 E72D D20BDA             JNC   :DA0B     Run 'SYNTAX ERROR' if no
066                                         number given
067 E730 C9                RET
068                 *
069                 ********************************
070                 * READ LINE NUMBER INTO EBUF *
071                 ********************************
072                 *
073                 * Exit: CY=0: No linenumber given. Error exit if
074                 *             linenumber is 0 or > #FFFF.
075                 *       CY=1: O.K.
076                 *       C,HL: Updated.
077                 *       B preserved, AFDE corrupted.
078                 *
079 E731 CDD2DD     L3E114  CALL  :DDD2     Get char from line, neglect
080                                         tab + space
081 E734 CD24C0             CALL  :C024     Input INT number to MACC
082 E737 D0                 RNC             Abort if no linenr given
083 E738 C5                 PUSH  B
084 E739 E7                 RST   4         Copy MACC into reg ABCD
085 E73A 15                 DATA  :15
086 E73B B0                 ORA   B
087 E73C C251E7             JNZ   :E751     Error exit if > #FFFF
088 E73F B1                 ORA   C
089 E740 B2                 ORA   D
090 E741 CA51E7             JZ    :E751     Error exit if nr = 0
091 E744 5A                 MOV   E,D       ) Linenr in DE
092 E745 51                 MOV   D,C       )
093 E746 72                 MOV   M,D       Hibyte linenr into EBUF
094 E747 CD18E0             CALL  :E018     Update EBUF pointer
095 E74A 73                 MOV   M,E       Lobyte linenr into EBUF
096 E74B CD18E0             CALL  :E018     Update EBUF pointer
097 E74E C1                 POP   B
098 E74F 37                 STC             CY=1 (O.K.)
099 E750 C9                 RET
100
101                 * Error exit:
102
103 E751 C1         L3E115  POP   B
104 E752 3E15               MVI   A,:15
105 E754 C3F5D9             JMP   :D9F5     Run error 'NUMBER OUT OF
106                                         RANGE'
107                 *
108                 ****************************************
109                 * ENCODE BINARY OPERATION INTO EBUF *
110                 ****************************************
111                 *
112                 * Entry: A : Result of 3E7CF:
113                 *            1xx xxxxx : compute type / opcode.
114                 *        E : Table code byte.
115                 *        HL: 1st free location EBUF.
116                 *
117 E757 F5         L3E116  PUSH  PSW
118 E758 7B                MOV   A,E       Get conversion code byte
119 E759 EB                XCHG
120 E75A 2A3B01            LHLD  :013B     Get addr last operator in
121                                         EBUF
122 E75D EB                XCHG            in DE
123 E75E CD70E7            CALL  :E770     Add conversion byte for
124                                         2nd operand
125 E761 0F                RRC
```

```
126 E762 0F                 RRC
127 E763 EB                 XCHG
128 E764 2A3901             LHLD  :0139     Get addr in EBUF for next
129                                         operator
130 E767 EB                 XCHG
131 E768 CD70E7             CALL  :E770     Add conversion byte for
132                                         1st operand
133 E76B F1                 POP   PSW       Restore compute/opcode in A
134 E76C CD83E7             CALL  :E783     Insert it into EBUF
135 E76F C9                 RET
136                *
137                ********************************************
138                * ADD INT/FPT CONVERSION BYTE TO EXPRESSION *
139                ********************************************
140                *
141                * Entry: A : Conversion byte:
142                *            #01: Convert FPT to INT.
143                *            #02: Convert INT to FPT.
144                *
145 E770 F5        L3E117  PUSH  PSW
146 E771 E603               ANI   :03       Conversion only
147 E773 CA81E7             JZ    :E781     Jump if no conversion reqd
148 E776 1F                 RAR             CY=1 if FPT to INT
149 E777 3E9F               MVI   A,:9F     Conv.byte FPT to INT
150 E779 DA7EE7             JC    :E77E
151 E77C 3EBF               MVI   A,:BF     Conv.byte INT to FPT
152 E77E CD83E7     L3E118  CALL  :E783     Insert conv.byte into EBUF
153 E781 F1         L3E119  POP   PSW
154 E782 C9                 RET
155                *
156                ***************************
157                * INSERT BYTE INTO EBUF *
158                ***************************
159                *
160                * Data is moved 1 byte to create space for byte
161                * to be inserted.
162                *
163                * Entry: A : Byte to be inserted.
164                *        DE: Startaddress source bank.
165                *        HL: Endaddress source bank +1.
166                * Exit:  HL= HL + 1.
167                *        ABCDE preserved.
168                *
169 E783 C5         L3E120  PUSH  B
170 E784 42                 MOV   B,D       ) Start source in BC
171 E785 4B                 MOV   C,E       )
172 E786 03                 INX   B         Destination 1 byte higher
173 E787 F5                 PUSH  PSW
174 E788 D5                 PUSH  D
175 E789 CD18E0             CALL  :E018     Update EBUF pointer
176 E78C E5                 PUSH  H
177 E78D 2B                 DCX   H
178 E78E CD4FDE             CALL  :DE4F     Move EBUF contents 1 byte
179 E791 E1                 POP   H
180 E792 D1                 POP   D
181 E793 F1                 POP   PSW
182 E794 12                 STAX  D         Store byte into EBUF
183 E795 C1                 POP   B
184 E796 C9                 RET
185                *
186                *
187                *
```

```
                        ****************************************
                        * ENCODE AN UNITARY OPERATOR FOR A TERM *
                        ****************************************
                        *
                        * Entry: A : Code according to table CFD8.
                        * Exit:  BCHL preserved. DE corrupted.
                        *        A : Code byte:
                        *                    +     -    INOT
                        *             INT    BC    BD    BE
                        *             FPT    9C    9D    *
                        *
E797 F5         L3E121  PUSH  PSW
E798 213601             LXI   H,:0136   Addr type last expression
E79B E61F               ANI   :1F       Opcode only
E79D FE00               CPI   :00       '+' ?
E79F 161C               MVI   D,:1C
E7A1 CABAE7             JZ    :E7BA     Then jump
E7A4 FE01               CPI   :01       '-' ?
E7A6 161D               MVI   D,:1D
E7A8 CABAE7             JZ    :E7BA     Then jump
E7AB FE1E               CPI   :1E       'INOT' ?
E7AD C20BDA             JNZ   :DA0B     Run 'SYNTAX ERROR' if not

                        * If 'INOT':

E7B0 7E                 MOV   A,M       Get type last expression
E7B1 FE10               CPI   :10       Must be INT
E7B3 C21ADA             JNZ   :DA1A     Run error 'TYPE MISMATCH'
                                        if not
E7B6 3EBE               MVI   A,:BE     Code 'INOT' in A
E7B8 E1         L3E122  POP   H
E7B9 C9                 RET

                        * If '+' or '-':

E7BA 7E         L3E123  MOV   A,M       Get type last expression
E7BB FE10               CPI   :10
E7BD 1EA0               MVI   E,:A0
E7BF CACAE7             JZ    :E7CA     Jump if INT
E7C2 7E                 MOV   A,M       Get type last expression
E7C3 FE00               CPI   :00
E7C5 1E80               MVI   E,:80
E7C7 C21ADA             JNZ   :DA1A     Run error 'TYPE MISMATCH'
                                        if not FPT
E7CA 7A         L3E124  MOV   A,D       ) Set up code in A
E7CB B3                 ORA   E         )
E7CC C3B8E7             JMP   :E7B8
                        *
                        ****************************************
                        * OBTAIN TYPE INFO FOR BINARY OPERATION *
                        ****************************************
                        *
                        * Entry: A : Code for binary operation (lower 5
                        *            bits).
                        *        E : Type 1st operand.
                        *     TYPE: Type 2nd operand.
                        *
                        * Routine compares both types. If different, one
                        * must be INT and the other FPT, else type mismatch
                        * error.
                        * Type conversion and operation type are obtained
                        * from table on 3E835. Type mismatch if illegal
```

```
250                 * type.
251                 * Type of result is stored in TYPE.
252                 *
253                 * Exit: E : Type code from table.
254                 *       A : Code for EBUF:  1xx xxxxx:
255                 *           bits 5,6: type of compute required:
256                 *                     0 : FPT
257                 *                     1 : INT
258                 *                     2 : STR
259                 *                     3 : Boolean
260                 *           bits 0-4: Opcode.
261                 *       BCDHL preserved.
262                 *
263 E7CF E5          L3E125  PUSH  H
264 E7D0 D5                  PUSH  D
265 E7D1 E61F                ANI   :1F       Opcode only
266 E7D3 F5                  PUSH  PSW
267 E7D4 CD1FE8              CALL  :E81F     Set D according to opcode
268 E7D7 3A3601              LDA   :0136     Get type latest expression
269 E7DA BB                  CMP   E         Compare both types
270 E7DB C209E8              JNZ   :E809     Jump if not identical
271 E7DE 07                  RLC             )
272 E7DF 07                  RLC             ) Type code from TYPE in
273 E7E0 07                  RLC             ) lonibble
274 E7E1 07                  RLC             )
275 E7E2 E603                ANI   :03       Only lower 2 bits
276 E7E4 6F                  MOV   L,A       in L
277 E7E5 7A          L3E126  MOV   A,D       Get opcode group (0-5)
278 E7E6 87                  ADD   A         *2
279 E7E7 57                  MOV   D,A       in D
280 E7E8 87                  ADD   A         *4
281 E7E9 82                  ADD   D         *6 (find group in table)
282 E7EA 85                  ADD   L         Find pos in grouptable
283 E7EB 2135E8              LXI   H,:E835   Startaddr result table
284 E7EE CD30DE              CALL  :DE30     Find addr resultcode in
285                                          table
286 E7F1 7E                  MOV   A,M       Get resultcode
287 E7F2 3C                  INR   A         Check if code is FF
288 E7F3 CA1ADA              JZ    :DA1A     Then run error 'TYPE
289                                          MISMATCH'
290 E7F6 3D                  DCR   A
291 E7F7 E630                ANI   :30       Get type of result only
292 E7F9 323601              STA   :0136     Store type latest expression
293 E7FC D1                  POP   D         Get code for binary
294                                          operation in D
295 E7FD 5E                  MOV   E,M       ) Get resultcode from table
296 E7FE 7E                  MOV   A,M       ) in E and in A
297 E7FF 1F                  RAR
298 E800 E660                ANI   :60       Reqd computing in bits 5,6
299 E802 B2                  ORA   D         Add opcode in bits 0-4
300 E803 F680                ORI   :80       Set bit 7
301 E805 E1                  POP   H
302 E806 54                  MOV   D,H       Restore D
303 E807 E1                  POP   H
304 E808 C9                  RET
305
306                 * If both types not identical:
307
308 E809 2E04        L3E127  MVI   L,:04
309 E80B FE00                CPI   :00       TYPE is FPT ?
310 E80D CA16E8              JZ    :E816     Then jump
311 E810 2C                  INR   L
```

```
312 E811 FE10                   CPI   :10       TYPE is INT ?
313 E813 C21ADA                 JNZ   :DA1A     Run error 'TYPE MISMATCH'
314                                             if not
315 E816 83             L3E128  ADD   E         Add other type
316 E817 FE10                   CPI   :10       Result must be #10
317 E819 C21ADA                 JNZ   :DA1A     Run error 'TYPE MISMATCH'
318                                             if not
319 E81C C3E5E7                 JMP   :E7E5     Calc conversion
320                     *
321                     * SET D DEPENDING ON OPCODE BINARY OPERATOR:
322                     *
323                     * Entry: A : Opcode binary operator (table #CF91).
324                     * Exit: ABCEHL preserved.
325                     *
326 E81F 1600            L3E129  MVI   D,:00     D=0
327 E821 FE01                   CPI   :01
328 E823 D8                     RC              Ready if opcode is 0 (+)
329 E824 14                     INR   D         D=1
330 E825 FE04                   CPI   :04
331 E827 D8                     RC              Ready if opcode is 1,2 or
332                                             3 (-,/,*)
333 E828 1602                   MVI   D,:02     D=2
334 E82A C8                     RZ              Ready if opcode is 4 (^)
335 E82B 14                     INR   D         D=3
336 E82C FE10                   CPI   :10
337 E82E D8                     RC              Ready if opcode is 5-F
                                                (IOR,IAND,IXOR,SHL,SHR,MOD)
338
339 E82F 14                     INR   D         D=4
340 E830 FE18                   CPI   :18
341 E832 D8                     RC              Ready if opcode is 10-17
                                                (>=, <=, >, <, =, <>)
342
343 E833 14                     INR   D         D=5
344 E834 C9                     RET             If opcode >= 18 (AND,OR)
345                     *
346                     * TABLE WITH TYPE RESULTS:
347                     *
348                     * The table gives the relation between input
349                     * operands, the binary operator and the result
350                     * for different groups of binary operations.
351                     * The groupnumber is calculated in 3E81F.
352                     *
353                     * Format each group: 6 bytes. Sequence:
354                     *             FPT/FPT
355                     *             INT/INT
356                     *             STR/STR
357                     *             LOGIC/LOGIC
358                     *             INT/FPT
359                     *             FPT/INT
360                     * Format each byte:
361                     *       bit 7,6: type arithmetic ) 0: FPT  1: INT
362                     *       bit 5,4: Type result     ) 2: STR  3: logic
363                     *       bit 3,2: Conversion left operand.
364                     *       bit 1,0: Conversion right operand.
365                     *                0 : No conversion.
366                     *                1 : Convert to INT.
367                     *                2 : Convert to FPT.
368                     *       FF : Not possible.
369                     *
370 E835 00              L3E385  DATA  :00       Group D=0:
371 E836 50                     DATA  :50       +
372 E837 A0                     DATA  :A0
373 E838 FF                     DATA  :FF
```

```
374 E839 08                     DATA  :08
375 E83A 02                     DATA  :02
376                   *
377 E83B 00                     DATA  :00       Group D=1:
378 E83C 50                     DATA  :50       -,/,*
379 E83D FF                     DATA  :FF
380 E83E FF                     DATA  :FF
381 E83F 08                     DATA  :08
382 E840 02                     DATA  :02
383                   *
384 E841 00                     DATA  :00       Group D=2:
385 E842 0A                     DATA  :0A       ^
386 E843 FF                     DATA  :FF
387 E844 FF                     DATA  :FF
388 E845 08                     DATA  :08
389 E846 02                     DATA  :02
390                   *
391 E847 55                     DATA  :55       Group D=3:
392 E848 50                     DATA  :50       IAND,IOR,IXOR,MOD,SHL,SHR
393 E849 FF                     DATA  :FF
394 E84A FF                     DATA  :FF
395 E84B 51                     DATA  :51
396 E84C 54                     DATA  :54
397                   *
398 E84D 30                     DATA  :30       Group D=4:
399 E84E 70                     DATA  :70       <,>,<>,=,<=,>=
400 E84F B0                     DATA  :B0
401 E850 FF                     DATA  :FF
402 E851 38                     DATA  :38
403 E852 32                     DATA  :32
404                   *
405 E853 FF                     DATA  :FF       Group D=5:
406 E854 FF                     DATA  :FF       AND,OR
407 E855 FF                     DATA  :FF
408 E856 F0                     DATA  :F0
409 E857 FF                     DATA  :FF
410 E858 FF                     DATA  :FF
411                   *
412                   *
413                   *
414 E859                        END
```

```
*****************************
* S Y M B O L   T A B L E *
*****************************

ELN     E72A    L3E110 E701    L3E111 E70A    L3E112 E727
L3E114 E731    L3E115 E751    L3E116 E757    L3E117 E770
L3E118 E77E    L3E119 E781    L3E120 E783    L3E121 E797
L3E122 E7B8    L3E123 E7BA    L3E124 E7CA    L3E125 E7CF
L3E126 E7E5    L3E127 E809    L3E128 E816    L3E129 E81F
L3E385 E835    RDID   E6FD
```

```
002                             ORG   :E859
003                     *
004                     *
005                     *
006                     ********************************
007                     * CHECK STATEMENT TERMINATOR *
008                     ********************************
009                     *
010                     * Get character from line and checks if it is
011                     * a correct terminator (':' or car.ret).
012                     *
013                     * Exit: Z=1: correct terminator.
014                     *       Z=0: incorrect.
015                     *       BCDEHL preserved. A corrupted.
016                     *
017 E859 CDD2DD         TSEOC   CALL  :DDD2     Get char from line, neglect
018                                             tab + space
019 E85C FE3A                   CPI   :3A       Is it ':' ?
020 E85E C8                     RZ
021 E85F FE0D                   CPI   :0D       Is it 'CR' ?
022 E861 C9                     RET
023                     *
024                     ************************************
025                     * CHECK IF NEXT CHARACTER IS ',' *
026                     ************************************
027                     *
028                     * Exit: C updated, AF corrupted, BDEHL preserved.
029                     *
030 E862 CD67E8         L3E131  CALL  :E867     Check if next char is ','
031 E865 2C                     DATA  :2C
032 E866 C9                     RET
033                     *
034                     ************************
035                     * CHECK NEXT CHARACTER *
036                     ************************
037                     *
038                     * Routine finds next valid character in input. If
039                     * it is not the character expected: syntax error.
040                     *
041                     * Entry: C : Points to input.
042                     *        ASCII-value of character to compare with
043                     *        on stack.
044                     * Exit:   If correct: C updated, AF corrupted,
045                     *        BDEHL preserved.
046                     *
047 E867 E3             ECHRI   XTHL            HL pnts to expected char
048 E868 CDD2DD                 CALL  :DDD2     Get char from line, neglect
049                                             tab + space
050 E86B BE                     CMP   M         Is it expected one ?
051 E86C C20BDA                 JNZ   :DA0B     Run 'SYNTAX ERROR' if not
052 E86F 0C                     INR   C         Pnts to next input
053 E870 23                     INX   H         )
054 E871 E3                     XTHL            ) Update SP
055 E872 C9                     RET
056                     *
057                     ********************************
058                     * ENCODE 'ERASE' - (not used) *
059                     ********************************
060                     *
061                     * The BASIC command 'ERASE' is cancelled.
062                     *
063 E873 1178E6         L3E133  LXI   D,:E678   Addr routine encode array
```

```
                                                  without arguments
E876 C32AE1             JMP   :E12A       Encode
                *
                ********************************
                * INPUT FPT NUMBER INTO MACC *
                ********************************
                *
                * Entry: Z=1: Change sign too.
                * Exit:  C updated, ABDEHL preserved.
                *         CY=0: Error.
                *
E879 CD1EC0     L3E134  CALL  :C01E       Input FPT number to MACC
E87C C0                 RNZ               Ready if Z=0
E87D E7                 RST   4           Else change sign MACC
E87E 1B                 DATA  :1B
E87F C9                 RET
                *
                *********************************
                * STORE QUOTED TEXT INTO EBUF *
                *********************************
                *
                * Entry: C points to 1st '"'.
                *
E880 0C         L3E135  INR   C
E881 C39CE6             JMP   :E69C       Store text in EBUF
                *
                ***********************************
                * STORE A HEX NUMBER INTO EBUF *
                ***********************************
                *
                * Entry: C points to '#' of hex number.
                *
E884 0C         EHEX    INR   C
E885 C390E5             JMP   :E590       Hex nr into EBUF
                *
                *************************************
                * ENCODE AN INT NUMBER INTO EBUF *
                *************************************
                *
E888 CDAFE8     L3E137  CALL  :E8AF       #14 into EBUF
E88B FE23               CPI   :23         '#' ?
E88D CA99E8             JZ    :E899       Then jump
E890 CD7BE5             CALL  :E57B       INT nr into EBUF
E893 D2B7E8     L3E139  JNC   :E8B7       Evt run 'SYNTAX ERROR'
E896 C35CE1             JMP   :E15C       Quit

                * If hex number:

E899 CD84E8     L3E138  CALL  :E884       Hex nr into EBUF
E89C C393E8             JMP   :E893
                *
E89F FF                 DATA  :FF
E8A0 FF                 DATA  :FF
E8A1 FF                 DATA  :FF
                *
                ******************
                * ENCODE 'DATA' *
                ******************
                *
E8A2 7D         EDATA   MOV   A,L
E8A3 FE42               CPI   :42
E8A5 C20BDA             JNZ   :DA0B       Run 'SYNTAX ERROR' if not
```

```
126 E8A8 C366E3              JMP   :E366     Encode text
127                      *
128                      ***********************************
129                      * part of END ENCODING (3E6B5) *
130                      ***********************************
131                      *
132 E8AB 1C              L3E140  INR   E
133 E8AC C3C8E6                 JMP   :E6C8
134                      *
135                      ************************************
136                      * CODE FOR INT NUMBER INTO EBUF *
137                      ************************************
138                      *
139                      * Gets also next character to encode.
140                      *
141 E8AF 3614            L3E141  MVI   M,:14     INT code (#14) in EBUF
142 E8B1 CD18E0                 CALL  :E018     Update EBUF pointer
143 E8B4 C3D2DD                 JMP   :DDD2     Get char from line, neglect
144                                             tab + space
145                      *
146                      ********************************************************
147                      * ERROR EXIT OF ENCODE INT NR INTO EBUF (3E888) *
148                      ********************************************************
149                      *
150 E8B7 2A3201          L3E142  LHLD  :0132     Get EFEPT
151 E8BA 11FCFF                 LXI   D,:FFFC
152 E8BD 19                     DAD   D         Set back linepntr
153 E8BE 220001                 SHLD  :0100     Store start current line
154 E8C1 C30BDA                 JMP   :DA0B     Run 'SYNTAX ERROR'
155                      *
156 E8C4 FF                     DATA  :FF
157                      *
158                      ****************************************
159                      * ASCII TABLE UPPER CASE (UNSHIFTED) *
160                      ****************************************
161                      *
162 E8C5 30              KEYTU   DATA  :30       0
163 E8C6 31                      DATA  :31       1
164 E8C7 32                      DATA  :32       2
165 E8C8 33                      DATA  :33       3
166 E8C9 34                      DATA  :34       4
167 E8CA 35                      DATA  :35       5
168 E8CB 36                      DATA  :36       6
169 E8CC 37                      DATA  :37       7
170 E8CD 38                      DATA  :38       8
171 E8CE 39                      DATA  :39       9
172 E8CF 3A                      DATA  :3A       :
173 E8D0 3B                      DATA  :3B       ;
174 E8D1 2C                      DATA  :2C       ,
175 E8D2 2D                      DATA  :2D       -
176 E8D3 2E                      DATA  :2E       .
177 E8D4 2F                      DATA  :2F       /
178 E8D5 0D                      DATA  :0D       car ret
179 E8D6 41                      DATA  :41       A
180 E8D7 42                      DATA  :42       B
181 E8D8 43                      DATA  :43       C
182 E8D9 44                      DATA  :44       D
183 E8DA 45                      DATA  :45       E
184 E8DB 46                      DATA  :46       F
185 E8DC 47                      DATA  :47       G
186 E8DD 48                      DATA  :48       H
187 E8DE 49                      DATA  :49       I
```

```
188 E8DF 4A                     DATA  :4A       J
189 E8E0 4B                     DATA  :4B       K
190 E8E1 4C                     DATA  :4C       L
191 E8E2 4D                     DATA  :4D       M
192 E8E3 4E                     DATA  :4E       N
193 E8E4 4F                     DATA  :4F       O
194 E8E5 50                     DATA  :50       P
195 E8E6 51                     DATA  :51       Q
196 E8E7 52                     DATA  :52       R
197 E8E8 53                     DATA  :53       S
198 E8E9 54                     DATA  :54       T
199 E8EA 55                     DATA  :55       U
200 E8EB 56                     DATA  :56       V
201 E8EC 57                     DATA  :57       W
202 E8ED 58                     DATA  :58       X
203 E8EE 59                     DATA  :59       Y
204 E8EF 5A                     DATA  :5A       Z
205 E8F0 5B                     DATA  :5B       [
206 E8F1 5E                     DATA  :5E       ^
207 E8F2 20                     DATA  :20       space
208 E8F3 00                     DATA  :00       (rept)
209 E8F4 08                     DATA  :08       char del
210 E8F5 10                     DATA  :10       cursor up
211 E8F6 11                     DATA  :11       cursor down
212 E8F7 12                     DATA  :12       cursor left
213 E8F8 13                     DATA  :13       cursor right
214 E8F9 09                     DATA  :09       tab
215 E8FA 80                     DATA  :80       ctrl
216 E8FB 00                     DATA  :00       (break)
217 E8FC 00                     DATA  :00       (shift)
218                     *
219                     ***************************************
220                     * ASCII TABLE LOWER CASE (SHIFTED) *
221                     ***************************************
222                     *
223 E8FD 30             KEYTS   DATA  :30       0
224 E8FE 21                     DATA  :21       !
225 E8FF 22                     DATA  :22       "
226 E900 23                     DATA  :23       #
227 E901 24                     DATA  :24       $
228 E902 25                     DATA  :25       %
229 E903 26                     DATA  :26       &
230 E904 27                     DATA  :27       '
231 E905 28                     DATA  :28       (
232 E906 29                     DATA  :29       )
233 E907 2A                     DATA  :2A       *
234 E908 2B                     DATA  :2B       +
235 E909 3C                     DATA  :3C       <
236 E90A 3D                     DATA  :3D       =
237 E90B 3E                     DATA  :3E       >
238 E90C 3F                     DATA  :3F       ?
239 E90D 0D                     DATA  :0D       car ret
240 E90E 61                     DATA  :61       a
241 E90F 62                     DATA  :62       b
242 E910 63                     DATA  :63       c
243 E911 64                     DATA  :64       d
244 E912 65                     DATA  :65       e
245 E913 66                     DATA  :66       f
246 E914 67                     DATA  :67       g
247 E915 68                     DATA  :68       h
248 E916 69                     DATA  :69       i
249 E917 6A                     DATA  :6A       j
```

```
250 E918 6B                 DATA  :6B       k
251 E919 6C                 DATA  :6C       l
252 E91A 6D                 DATA  :6D       m
253 E91B 6E                 DATA  :6E       n
254 E91C 6F                 DATA  :6F       o
255 E91D 70                 DATA  :70       p
256 E91E 71                 DATA  :71       q
257 E91F 72                 DATA  :72       r
258 E920 73                 DATA  :73       s
259 E921 74                 DATA  :74       t
260 E922 75                 DATA  :75       u
261 E923 76                 DATA  :76       v
262 E924 77                 DATA  :77       w
263 E925 78                 DATA  :78       x
264 E926 79                 DATA  :79       y
265 E927 7A                 DATA  :7A       z
266 E928 5D                 DATA  :5D       ]
267 E929 7E                 DATA  :7E       ~
268 E92A 20                 DATA  :20       space
269 E92B 00                 DATA  :00       (rept)
270 E92C 08                 DATA  :08       char del
271 E92D 14                 DATA  :14       window up
272 E92E 15                 DATA  :15       window down
273 E92F 16                 DATA  :16       window left
274 E930 17                 DATA  :17       window right
275 E931 09                 DATA  :09       tab
276 E932 80                 DATA  :80       ctrl
277 E933 00                 DATA  :00       (break)
278 E934 00                 DATA  :00       (shift)
279                     *
280                     ***************************************
281                     * GET INPUTS FROM KEYBOARD OR DINC *
282                     ***************************************
283                     *
284                     * Part of RESET (C719). Determines input source
285                     * depending on 1st input done.
286                     *
287 E935 CDBBD6     L3E143  CALL  :D6BB     Scan keyb; char in A
288 E938 D8                 RC              Ready if break pressed
289 E939 C0                 RNZ             Ready if key input done
290 E93A C3F4EF             JMP   :EFF4     Else: Get input from DINC
291                     *
292 E93D FF                 DATA  :FF
293 E93E FF                 DATA  :FF
294                     *
295                     ************************************************
296                     * LOAD ASCII VALUE FOR KEY PRESSED IN BUFFER *
297                     ************************************************
298                     *
299                     * From the key pressed, the offset to the start-
300                     * address of the ASCII table is calculated. The
301                     * ASCII value for the pressed key is stored in
302                     * the circular buffer KLIND.
303                     *
304                     * Entry: B : Column number.
305                     *        C : Row number.
306                     * Exit:  All registers preserved.
307                     *
308 E93F F5          INKEY   PUSH  PSW
309 E940 C5                 PUSH  B
310 E941 E5                 PUSH  H
311 E942 3E07               MVI   A,:07     )
```

```
312 E944 90                  SUB   B          )
313 E945 87                  ADD   A          ) Calc offset of startaddr
314 E946 87                  ADD   A          ) for key pressed.
315 E947 87                  ADD   A          )   Store it in C
316 E948 81                  ADD   C          )
317 E949 4F                  MOV   C,A        )
318 E94A 2AA702              LHLD  :02A7      Get startaddr ASCII table
319 E94D 0600                MVI   B,:00
320 E94F FE11                CPI   :11        )
321 E951 DA5DE9              JC    :E95D      ) Check if key is a char
322 E954 FE2B                CPI   :2B        ) A-Z
323 E956 D25DE9              JNC   :E95D      )
324 E959 3AC302              LDA   :02C3      Get shift lock value
325 E95C 47                  MOV   B,A        in B
326 E95D 3AB002      L3E145  LDA   :02B0      Get 'shift' byte
327 E960 A8                  XRA   B          Take CTRL into account
328 E961 E640                ANI   :40        A=#40 if shift, 00 when not
329 E963 CA6BE9              JZ    :E96B      Jump if no shift
330 E966 D5                  PUSH  D
331 E967 113800              LXI   D,:0038    Add. offset for lower case
332                                           table
333 E96A 19                  DAD   D          Startaddr lower case table
334                                           now in HL
335 E96B D1                  POP   D
336 E96C 0600        L3E146  MVI   B,:00
337 E96E 09                  DAD   B          Add offset to startaddr
338 E96F 7E                  MOV   A,M        Get ASCII value from table
339 E970 B7                  ORA   A          Check if Break, Rept, Shift
340 E971 CA8EE9              JZ    :E98E      Then Pop, ret
341 E974 FE80                CPI   :80        Check if CTRL
342 E976 CA92E9              JZ    :E992      Then update CTRL flag
343 E979 47                  MOV   B,A        Store ASCII value in B
344 E97A 2ABE02              LHLD  :02BE      Get addr next pos in KLIND
345 E97D E5                  PUSH  H          Store KLIIN on stack
346 E97E CD9CD6              CALL  :D69C      Update KLIND pointer
347 E981 3AC002              LDA   :02C0      Get lsbyte next output pos
348                                           of KLIND
349 E984 BD                  CMP   L          Compare with KLIIN
350 E985 CA8DE9              JZ    :E98D      Abort if buffer full
351 E988 22BE02              SHLD  :02BE      Update KLIIN
352 E98B E3                  XTHL             Get old KLIIN from stack
353 E98C 70                  MOV   M,B        Store ASCII char in KLIND
354 E98D E1          L3E147  POP   H
355 E98E E1          L3E148  POP   H
356 E98F C1                  POP   B
357 E990 F1                  POP   PSW
358 E991 C9                  RET
359
360                  * Update CTRL flag:
361
362 E992 3AC302      L3E149  LDA   :02C3      Get shiftlock value
363 E995 2F                  CMA              Invert it
364 E996 32C302              STA   :02C3      And store it again
365 E999 C38EE9              JMP   :E98E      Pop, ret
366                  *
367                  ****************
368                  * HEAP REQUEST *
369                  ****************
370                  *
371                  * The routine checks the heap for free areas. Evt.
372                  * consecutive free areas are consolidated. If this
373                  * procedure finds a free area min. 2 bytes larger
```

```
374                     * than requested, then it is reserved by setting
375                     * the length bytes (msb=0). An evt. resting free
376                     * area is set with length bytes and msb=1 (this
377                     * area must be >= 2 bytes).
378                     * The heap contents is never moved to obtain one
379                     * large consolidated area of free bytes!!
380                     *
381                     * Entry: DE: Length requested heap space.
382                     * Exit:  AFBCDE preserved.
383                     *        HL: Points to a 2-byte length of the re-
384                     *            quested gap. If no space available,
385                     *            it points to an error routine.
386                     *
387 E99C F5            HREQ    PUSH  PSW
388 E99D C5                    PUSH  B
389 E99E D5                    PUSH  D
390 E99F 42                    MOV   B,D       ) Reqd length in BC
391 E9A0 4B                    MOV   C,E       )
392 E9A1 2A9B02              HRQ10 LHLD  :029B     Get startaddr Heap
393 E9A4 56                    MOV   D,M       )
394 E9A5 23                    INX   H         ) Contents 1st 2 bytes of
395 E9A6 5E                    MOV   E,M       ) Heap in DE (length)
396 E9A7 23                    INX   H         )
397 E9A8 7A                    MOV   A,D       1st byte in A
398 E9A9 E67F                  ANI   :7F       Mask bit 'free/used'
399 E9AB BA                    CMP   D
400 E9AC 57                    MOV   D,A       D is length without msb
401 E9AD CA27D2                JZ    :D227     If area not free: check if
402                                              end of heap reached. JMP
403                                              #E9FA if not
404
405                     * If free area found: Check all next heap entries
406                     * and accumulate all free areas in succession:
407
408 E9B0 E5            HRQ20   PUSH  H         Save startaddr area +2
409 E9B1 19                    DAD   D         Begin next area in HL
410 E9B2 7E                    MOV   A,M
411 E9B3 B7                    ORA   A         Check msb of this area
412 E9B4 F2C7E9                JP    :E9C7     Jump if area occupied
413 E9B7 23                    INX   H
414 E9B8 7B                    MOV   A,E
415 E9B9 86                    ADD   M         Add lobyte length next area
416                                              length previous area
417 E9BA 5F                    MOV   E,A
418 E9BB 7A                    MOV   A,D
419 E9BC 2B                    DCX   H
420 E9BD 8E                    ADC   M         Add hibyte length next area
421                                              length previous area
422 E9BE E67F                  ANI   :7F       Skip bit 'free/occupied'
423 E9C0 57                    MOV   D,A
424 E9C1 13                    INX   D
425 E9C2 13                    INX   D         Add 2 extra bytes
426 E9C3 E1                    POP   H         Restore start free area +2
427 E9C4 C3B0E9                JMP   :E9B0     Check next area
428
429                     * Next area is not free:
430
431 E9C7 E1            HRQ30   POP   H         Restore start free area +2
432 E9C8 E5                    PUSH  H
433 E9C9 2B                    DCX   H
434 E9CA 73                    MOV   M,E       )
435 E9CB 2B                    DCX   H         ) Store total free length in
```

```
436 E9CC 7A                 MOV   A,D       ) 1st 2 bytes of free area
437 E9CD F680               ORI   :80       )
438 E9CF 77                 MOV   M,A       )
439                                         Space available here: HL pnt
440                                         to start free area +2; DE is
441                                         size free area; BC size reqd
442 E9D0 7B                 MOV   A,E       )
443 E9D1 91                 SUB   C         )
444 E9D2 6F                 MOV   L,A       ) Calc free length - reqd.
445 E9D3 7A                 MOV   A,D       ) length; result in HL
446 E9D4 98                 SBB   B         )
447 E9D5 67                 MOV   H,A       )
448 E9D6 FAF9E9             JM    :E9F9     Space not sufficient: Leave
449                                         consolidated area as free
450 E9D9 C2E4E9             JNZ   :E9E4     Jump if sufficient space
451 E9DC B5                 ORA   L
452 E9DD CAF0E9             JZ    :E9F0     Jump if just enough
453 E9E0 3D                 DCR   A
454 E9E1 CAF9E9             JZ    :E9F9     Not useable if 1 free byte
455                                         left
456
457                 * Set not used part of free area to free:
458
459 E9E4 EB          HR040  XCHG             Addr free area in DE
460 E9E5 1B                 DCX   D
461 E9E6 1B                 DCX   D         Reserve 2 bytes for length
462 E9E7 E1                 POP   H         Restore start free area +2
463 E9E8 E5                 PUSH  H
464 E9E9 09                 DAD   B         Add reqd length to find
465                                         start of resting free area
466 E9EA 7A                 MOV   A,D
467 E9EB F680               ORI   :80       Set free flag
468 E9ED 77                 MOV   M,A       )
469 E9EE 23                 INX   H         ) Length free area into heap
470 E9EF 73                 MOV   M,E       )
471
472                 * Reserve area for requested entry:
473
474 E9F0 E1          HR050  POP   H         Restore start free area +2
475 E9F1 2B                 DCX   H
476 E9F2 71                 MOV   M,C       )
477 E9F3 2B                 DCX   H         ) Reqd length in 1st 2 bytes
478 E9F4 70                 MOV   M,B       )
479 E9F5 D1                 POP   D
480 E9F6 C1                 POP   B
481 E9F7 F1                 POP   PSW
482 E9F8 C9                 RET
483
484                 * Area too small:
485
486 E9F9 E1          HR070  POP   H         Restore start free area
487 E9FA 19          HR075  DAD   D         HL pnts to next area
488 E9FB C3A4E9             JMP   :E9A4     Check next area
489                 *
490 E9FE FF                 DATA  :FF
491 E9FF FF                 DATA  :FF
492                 *
493                 *
494                 *
495                 *
496 EA00                    END
```

****
* S Y M B O L T A B L E *
****

ECHRI	E867	EDATA	E8A2	EHEX	E884	HREQ	E99C
HRQ10	E9A4	HRQ20	E9B0	HRQ30	E9C7	HRQ40	E9E4
HRQ50	E9F0	HRQ70	E9F9	HRQ75	E9FA	INKEY	E93F
KEYTS	E8FD	KEYTU	E8C5	L3E131	E862	L3E133	E873
L3E134	E879	L3E135	E880	L3E137	E888	L3E138	E899
L3E139	E893	L3E140	E8AB	L3E141	E8AF	L3E142	E8B7
L3E143	E935	L3E145	E95D	L3E146	E96C	L3E147	E98D
L3E148	E98E	L3E149	E992	TSEOC	E859		

```
002                             ORG   :EA00
003                     *
004                     *
005                     *
006                     *  ================
007                     *** UTILITY PACKAGE ***
008                     *  ================
009                     *
010                     **************
011                     * (not used) *
012                     **************
013                     *
014 EA00 E1             L3E158  POP   H
015 EA01 C309EA                 JMP   :EA09
016                     *
017                     ***********************
018                     * RETURN AFTER 'GO' *
019                     ***********************
020                     *
021 EA04 E5             L3E159  PUSH  H           Returnaddr after 'GO'
022 EA05 21DCBA                 LXI   H,:BADC     Dummy 'saved PC' to prevent
023                                               continuation 'G' with start
024                                               address
025 EA08 E3                     XTHL              Returnaddr in HL
026 EA09 225900         L3E160  SHLD  :0059       Save HL
027 EA0C E1                     POP   H           in HL: #BADC
028                                               Into initialisation
029                     *
030                     ************************
031                     * INITIALISE UTILITY *
032                     ************************
033                     *
034                     * CPU registers are saved in the utility work
035                     * area. Input from the keyboard is awaited.
036                     *
037                     * The address EA42 is the general return address
038                     * for all Utility commands.
039                     *
040 EA0D 225D00         CALRX   SHLD  :005D       Save PC (next instr)
041 EA10 F5                     PUSH  PSW
042 EA11 E1                     POP   H
043 EA12 225300                 SHLD  :0053       Save PSW
044 EA15 210000                 LXI   H,:0000
045 EA18 39                     DAD   SP
046 EA19 225B00                 SHLD  :005B       Save SP
047 EA1C EB                     XCHG
048 EA1D 223900                 SHLD  0057        Save DE
049 EA20 60                     MOV   H,B
050 EA21 69                     MOV   L,C
051 EA22 225500                 SHLD  :0055       Save BC
052 EA25 CDE7ED                 CALL  :EDE7       Invert nibbles in CPU
053                                               reg save area
054 EA28 2A5D00                 LHLD  :005D       Get addr next instr
055 EA2B 7C                     MOV   A,H
056 EA2C B5                     ORA   L
057 EA2D CA3CEA                 JZ    :EA3C       If it is 0000 (entry from
058                                               BASIC)
059                                               Else:
060 EA30 2B                     DCX   H           HL on addr current instr
061 EA31 7E                     MOV   A,M         Get opcode in A
062 EA32 E6C7                   ANI   :C7
063 EA34 FEC7                   CPI   :C7
```

```
064 EA36 C23CEA              JNZ   :EA3C     Jump if instr is a RST
065 EA39 225D00              SHLD  :005D     Save addr next instr
066 EA3C 217DEA      L3E162  LXI   H,:EA7D   Startaddr string table
067 EA3F CD2FED              CALL  :ED2F     Print 'PC UTILITY V3.3'
068
069                  * UT command look-up:
070
071 EA42 CD3AED      L3E163  CALL  :ED3A     Print car.ret
072 EA45 0E3E                MVI   C,:3E
073 EA47 CDB4EE              CALL  :EEB4     Print '>'
074 EA4A CD06ED              CALL  :ED06     Get keyb input, print char
075 EA4D 2142EA              LXI   H,:EA42   Returnaddr in HL
076 EA50 E5                  PUSH  H         Save it on stack
077 EA51 218DEA              LXI   H,:EA8D   Startaddr table commands
078 EA54 23          L3E164  INX   H
079 EA55 BE                  CMP   M         Compare input with table
080 EA56 DA62EA              JC    :EA62     Error if invalid input
081 EA59 23                  INX   H         )
082 EA5A 5E                  MOV   E,M       ) Get pointer to address
083 EA5B 23                  INX   H         ) of part. routine in DE
084 EA5C 56                  MOV   D,M       )
085 EA5D C254EA              JNZ   :EA54     Check with next command
086 EA60 EB                  XCHG            Startaddr routine in HL
087 EA61 E9                  PCHL            Go to this routine
088                  *
089                  *********
090                  * ERROR *
091                  *********
092                  *
093                  * The only error message in utility is '?'.
094                  *
095                  * Exit: B preserved, AFCDEHL corrupted.
096                  *
097 EA62 CD3AED      ERROR   CALL  :ED3A     Print car.ret
098 EA65 0E3F                MVI   C,:3F
099 EA67 CDB4EE              CALL  :EEB4     Print '?'
100 EA6A 2A5B00      ERRST   LHLD  :005B     Get saved SP
101 EA6D F9                  SPHL            Restore stackpointer
102 EA6E 00                  NOP
103 EA6F 00                  NOP
104 EA70 00                  NOP
105 EA71 C342EA              JMP   :EA42     Start again for new input
106                  *
107                  **********************
108                  * ENTRY FROM BASIC *
109                  **********************
110                  *
111 EA74 225900      RESET   SHLD  :0059     Save HL
112 EA77 210000              LXI   H,:0000   Addr next instr (dummy)
113 EA7A C30DEA              JMP   :EA0D     Init utility
114                  *
115                  ***************************
116                  * UTILITY SCREEN HEADER *
117                  ***************************
118                  *
119                  * 0C is clear screen; 20 is space.
120                  *
121 EA7D 0C          MSGS0   DATA  :0C
122 EA7E 50                  DATA  :50       P
123 EA7F 43                  DATA  :43       C
124 EA80 20                  DATA  :20
125 EA81 55                  DATA  :55       U
```

```
126 EA82 54                     DATA  :54       T
127 EA83 49                     DATA  :49       I
128 EA84 4C                     DATA  :4C       L
129 EA85 49                     DATA  :49       I
130 EA86 54                     DATA  :54       T
131 EA87 59                     DATA  :59       Y
132 EA88 20                     DATA  :20
133 EA89 56                     DATA  :56       V
134 EA8A 33                     DATA  :33       3
135 EA8B 2E                     DATA  :2E       .
136 EA8C 33                     DATA  :33       3
137                     *
138 EA8D 00                     DATA  :00       End table
139                     *
140                     ********************************
141                     * TABLE WITH UTILITY COMMANDS *
142                     ********************************
143                     *
144 EA8E 42             CMDTB  DATA  :42       B
145 EA8F A0C7                  DBL   :C7A0     return addr to Basic
146                     *
147 EA91 44                     DATA  :44       D
148 EA92 B3EA                   DBL   :EAB3     startaddr Display
149                     *
150 EA94 46                     DATA  :46       F
151 EA95 48ED                   DBL   :ED48     startaddr Fill
152                     *
153 EA97 47                     DATA  :47       G
154 EA98 8AED                   DBL   :ED8A     startaddr Go
155                     *
156 EA9A 4C                     DATA  :4C       L
157 EA9B 26EB                   DBL   :EB26     startaddr Look
158                     *
159 EA9D 4D                     DATA  :4D       M
160 EA9E 83EC                   DBL   :EC83     startaddr Move
161                     *
162 EAA0 52                     DATA  :52       R
163 EAA1 0FEF                   DBL   :EF0F     startaddr Read
164                     *
165 EAA3 53                     DATA  :53       S
166 EAA4 5CED                   DBL   :ED5C     startaddr Substitute
167                     *
168 EAA6 56                     DATA  :56       V
169 EAA7 77ED                   DBL   :ED77     startaddr Vector Examine
170                     *
171 EAA9 57                     DATA  :57       W
172 EAAA E4EE                   DBL   :EEE4     startaddr Write
173                     *
174 EAAC 58                     DATA  :58       X
175 EAAD 6EED                   DBL   :ED6E     startaddr Examine
176                     *
177 EAAF 5A                     DATA  :5A       Z
178 EAB0 BAEC                   DBL   :ECBA     startaddr Reset
179                     *
180 EAB2 FF                     DATA  :FF       End table
181                     *
182                     ***************
183                     * D - DISPLAY *
184                     ***************
185                     *
186                     * Displays contents of memory. Two address values
187                     * are required which specify the range of memory
```

```
                        * to be displayed. Break will abort the print out.
                        *
                        * Exit: CY=1, All registers corrupted.
                        *
EAB3 0E02               DISPK   MVI   C,:02      Nr of addr inputs required
EAB5 CDDEEA                     CALL  :EADE      Get laddr and haddr on stack
EAB8 0D                         DCR   C
EAB9 F262EA                     JP    :EA62      Error if only 1 addr given
EABC D1                         POP   D          Haddr in DE
EABD E1                         POP   H          Laddr in HL
                        *
                        * Direct call entry:
                        *
                        DISL1
EABE CD3AED             DISP    CALL  :ED3A      Print car.ret
EAC1 CD18ED                     CALL  :ED18      Print laddr in ASCII
EAC4 CD01ED             DISL2   CALL  :ED01      Print space
EAC7 7E                         MOV   A,M        Get contents laddr
EAC8 CD1DED                     CALL  :ED1D      Print it in ASCII
EACB CD80ED                     CALL  :ED80      INX H; check if ready
EACE D8                         RC               Quit if ready
EACF CDC2EE                     CALL  :EEC2      Scan for Break pressed,
                                                 abort if pressed.
EAD2 7D                         MOV   A,L
EAD3 E60F                       ANI   :0F        Last instr on line?
EAD5 CABEEA                     JZ    :EABE      Then car.ret and continue
EAD8 C3C4EA                     JMP   :EAC4      Next addr
                        *
                        ***************************
                        * ADDRESS ARGUMENT INPUT *
                        ***************************
                        *
                        * The keyboard is scanned and the inputs are
                        * evaluated.
                        * (C) address arguments will be input and put on
                        * stack (LIFO). On return, C contains the
                        * difference between number entered and number
                        * desired. At least one argument is returned.
                        * Only the last 4 hex characters are used for the
                        * address value. Arguments are delimited by a
                        * space. The entry is terminated by CR, ESC, last
                        * argument or an invalid character (then error
                        * exit). Escape returns with CY set.
                        *
                        * Entry: C: max. nr of datablocks/addresses.
                        * Exit:  B: Last character typed in (terminator).
                        *        AFHL corrupted, DE preserved.
                        *
EADB AF                 ADALT   XRA   A          A=0
EADC B9                         CMP   C          Max nr of inputs reached?
EADD C8                         RZ               Then return
EADE 210000             ADARG   LXI   H,:0000
EAE1 CD06ED             ADACL   CALL  :ED06      Scan keyb, print char
EAE4 47                         MOV   B,A        Store char in B
EAE5 CD15EB                     CALL  :EB15      Convert it to hex
EAE8 DAF4EA                     JC    :EAF4      If char not 0-F: check if
                                                 delimiter
EAEB 29                 ADACE   DAD   H          )
EAEC 29                         DAD   H          ) Move bits 1 nibble
EAED 29                         DAD   H          ) (compose addr from inputs)
EAEE 29                         DAD   H          )
EAEF 85                         ADD   L          Add hex char to L
```

```
EAF0 6F              MOV   L,A       and store it
EAF1 C3E1EA          JMP   :EAE1     Next char

                * Check for delimiter/terminator:

EAF4 E3         ADADC   XTHL         Save given addr on stack
EAF5 E5                 PUSH  H      Save returnaddr again
EAF6 0D                 DCR   C      decr input counter
EAF7 78                 MOV   A,B    Get char last input
EAF8 FE20               CPI   :20    Space ?
EAFA CADBEA             JZ    :EADB  Then get next addr
EAFD FE0D               CPI   :0D    Car.ret ?
EAFF C8                 RZ           Then ready
EB00 FE12               CPI   :12    Escape?
EB02 37                 STC          Then CY=1
EB03 C8                 RZ           and return
EB04 C362EA             JMP   :EA62  Else goto Error

                * As ADARG, but carry return if 1st character is
                * not a legal hex digit:

EB07 210000     ADART   LXI   H,:0000  Immediate delimiter
EB0A CD06ED             CALL  :ED06    Scan keyb, print char
EB0D 47                 MOV   B,A      Store char in B
EB0E CD15EB             CALL  :EB15    Convert it to hex
EB11 D8                 RC             Return if no hex char
EB12 C3EBEA             JMP   :EAEB    Into previous routine
                *
                * CONVERT NUMBER FROM ASCII TO HEX-VALUE:
                *
                * Entry: Character in A (bit 7 must be 0).
                * Exit:   CY=0: Hex-value in A.
                *         CY=1: Input was not 0-F; (A) useless.
                *         BCDEHL preserved.
                *
EB15 D630       ASHEX   SUI   :30
EB17 D8                 RC             Error if #00-#2F
EB18 FE0A               CPI   :0A
EB1A DA24EB             JC    :EB24    O.K. if number 0-9
EB1D D607               SUI   :07
EB1F FE0A               CPI   :0A
EB21 D8                 RC             Error if #3A-#3F
EB22 FE10               CPI   :10
EB24 3F         ASHCC   CMC            O.K. if 0-9, A-F
EB25 C9                 RET
                *
                *
                *
EB26                    END
```

```
****************************
* S Y M B O L   T A B L E *
****************************

ADACE  EAEB   ADACL  EAE1   ADADC  EAF4   ADALT  EADB
ADARG  EADE   ADART  EB07   ASHCC  EB24   ASHEX  EB15
CALRX  EA0D   CMDTB  EA8E   DISL1  EABE   DISL2  EAC4
DISP   EABE   DISPK  EAB3   ERROR  EA62   ERRST  EA6A
L3E158 EA00   L3E159 EA04   L3E160 EA09   L3E162 EA3C
L3E163 EA42   L3E164 EA54   MSGSO  EA7D   RESET  EA74
```

```
002                             ORG   :EB26
003                     *
004                     *
005                     *
006                     ************
007                     * L - LOOK *
008                     ************
009                     *
010 EB26 0E03           LOOKK   MVI   C,:03     Nr datablocks allowed
011 EB28 CD07EB                 CALL  :EB07     Scan keyb, display char,
012                                             get addresses on stack
013 EB2B 2A5D00                LHLD  :005D     Get addr next instr
014 EB2E EB                     XCHG            in DE
015 EB2F 79                     MOV   A,C       Get nr of inputs done
016 EB30 FE03                   CPI   :03       No addr given?
017 EB32 3E00                   MVI   A,:00
018 EB34 CC56EB                 CZ    :EB56     No addr: Check CR given
019 EB37 C441EB                 CNZ   :EB41     Else: store windows and
020                                             start program
021 EB3A 325000                 STA   :0050     Set Look flag
022 EB3D EB                     XCHG            Addr next instr in HL
023 EB3E C335EF                 JMP   :EF35     Init RST 0
024                     *
025                     * SET LOOK WINDOWS, START LOOK:
026                     *
027                     * Entry: A=0, C=3 minus number of fields read.
028                     * Exit:  Input 2 fields: A,C = 0.
029                     *        Input 3 fields: A,C = FF. DE corrupted.
030                     *        B preserved, HL corrupted.
031                     *
032 EB41 0D             L3E178  DCR   C
033 EB42 0D                     DCR   C
034 EB43 CA62EA                 JZ    :EA62     Error if only 1 addr given
035 EB46 E1                     POP   H         Get returnaddr from stack
036 EB47 E3                     XTHL            haddr window in HL
037 EB48 224800                 SHLD  :0048     Save haddr
038 EB4B E1                     POP   H         Return addr from stack
039 EB4C E3                     XTHL            laddr window in HL
040 EB4D 224A00                 SHLD  :004A     Save laddr
041 EB50 0C                     INR   C
042 EB51 C8                     RZ              Ready if only window given
043
044                     * If startaddress given:
045
046 EB52 3D                     DCR   A         A=FF
047 EB53 E1                     POP   H         Get returnaddr from stack
048 EB54 D1                     POP   D         Get startaddr program in DE
049 EB55 E9                     PCHL            Return
050                     *
051                     ********************************
052                     * GO/LOOK: CHECK FOR CAR.RET *
053                     ********************************
054                     *
055                     * Entry: B: Character to be checked.
056                     * Exit:  AF corrupted, BCDEHL preserved.
057                     *
058 EB56 78             L3E179  MOV   A,B       Get last char typed in
059 EB57 D60D                   SUI   :0D
060 EB59 C262EA                 JNZ   :EA62     Error if not CR
061 EB5C C9                     RET
062                     *
063                     *
```

```
064                     **********************
065                     * RESTART 0 (RST 0) *
066                     **********************
067                     *
068                     * The RST 0 function is used to operate 'LOOK'.
069                     * Via a timer 1 interrupt, RST 0 is called.
070                     * The vectoraddress #EB5D must have been
071                     * initialised by a Z2 or Z3 command.
072                     *
073                     * On entry, HL and PC are on stack (see 0000
074                     * -0007).
075                     *
076 EB5D F5              L3E180  PUSH  PSW
077 EB5E 00                      NOP
078 EB5F 3A4700                 LDA   :0047     Get stored EI/DI
079 EB62 D6FB                   SUI   :FB
080 EB64 CA6CEB                 JZ    :EB6C     Jump if EI stored
081 EB67 3E01                   MVI   A,:01     Else: Set int. mask
082 EB69 32F8FF                 STA   :FFF8     for timer 1 only
083
084                     * Save all registers in RAM area:
085
086 EB6C FB              L3E181  EI
087 EB6D F1                     POP   PSW       Restore PSW
088 EB6E E1                     POP   H         Restore HL
089 EB6F 225900                 SHLD  :0059     Save HL
090 EB72 F5                     PUSH  PSW
091 EB73 C5                     PUSH  B
092 EB74 D5                     PUSH  D
093 EB75 E1                     POP   H
094 EB76 225700                 SHLD  :0057     Save DE
095 EB79 E1                     POP   H
096 EB7A 225500                 SHLD  :0055     Save BC
097 EB7D E1                     POP   H
098 EB7E 225300                 SHLD  :0053     Save PSW
099 EB81 E1                     POP   H
100 EB82 225D00                 SHLD  :005D     Save PC
101
102                     * Check if PC points to 004D-4E:
103
104 EB85 11B3FF                 LXI   D,:FFB3
105 EB88 19                     DAD   D
106 EB89 EB                     XCHG            DE=PC+FFB3 (=PC-004D)
107 EB8A 2A5100                 LHLD  :0051     Get addr where to continue
108 EB8D 7A                     MOV   A,D
109 EB8E B7                     ORA   A
110 EB8F C298EB                 JNZ   :EB98     Jump if out of interrupt
111                                             routine
112 EB92 B3                     ORA   E
113 EB93 FE04                   CPI   :04
114 EB95 DA3EEC                 JC    :EC3E     Jump if PC = 004D-4E
115
116                     * Put returnaddress on stack if current instruction
117                     * is a RST or a CALL instruction:
118
119 EB98 06C7            L3E182  MVI   B,:C7
120 EB9A 7E                     MOV   A,M       Get instr code of next instr
121 EB9B FECD                   CPI   :CD
122 EB9D CAACEB                 JZ    :EBAC     Jump if it is a CALL
123 EBA0 A0                     ANA   B         ) Check if it is a RST
124 EBA1 B8                     CMP   B         )
125 EBA2 CAAEEB                 JZ    :EBAE     Then jump
```

```
126 EBA5 7E                 MOV   A,M        Get instr code
127 EBA6 A0                 ANA   B          ) Check if it is a
128 EBA7 FEC4               CPI   :C4        ) conditional CALL
129 EBA9 C2B0EB             JNZ   :EBB0      Jump if not
130 EBAC 23         L3E183  INX   H
131 EBAD 23                 INX   H
132 EBAE 23         L3E184  INX   H
133 EBAF E3                 XTHL             Addr next instr on stack
134                                          if it was a RST or CALL
135
136                 * Check if current instruction is inside window:
137
138 EBB0 210000      L3E185 LXI   H,:0000
139 EBB3 39                 DAD   SP
140 EBB4 225B00             SHLD  :005B      Save SP
141 EBB7 CDE7ED             CALL  :EDE7      Exchange bytes in reg. save
142                                          area
143 EBBA 2A5100             LHLD  :0051      Get addr current instr
144 EBBD EB                 XCHG             in DE
145 EBBE 2A4A00             LHLD  :004A      Get laddr window
146 EBC1 CD45EC             CALL  :EC45      Compare DE-HL
147 EBC4 FADCEB             JM    :EBDC      Jump if addr outside window
148 EBC7 2A4800             LHLD  :0048      Get haddr window
149 EBCA CD45EC             CALL  :EC45      Compare DE-HL
150 EBCD D2DCEB             JNC   :EBDC      Jump if addr outside window
151
152                 * Print registers contents if address inside window:
153
154 EBD0 CD3AED             CALL  :ED3A      Print car.ret
155 EBD3 115100             LXI   D,:0051    Startaddr reg. save area
156 EBD6 219CEE             LXI   H,:EE9C    Startaddr symbol table
157 EBD9 CD44EE             CALL  :EE44      Print reg. contents
158 EBDC CDC2EE     L3E186  CALL  :EEC2      Scan for Break pressed;
159                                          evt run Break
160 EBDF 2A5D00             LHLD  :005D      Get addr next instr
161
162                 * Disable UT to trace itself:
163                 * Entry from init, RST0.
164                 * HL is address where to continu.
165
166 EBE2 3EEA       L3E187  MVI   A,:EA
167 EBE4 BC                 CMP   H          Check if hibyte = EA
168 EBE5 C2EEEB             JNZ   :EBEE      Jump if not
169 EBE8 3E0D               MVI   A,:0D
170 EBEA BD                 CMP   L          Check if lobyte = 0D
171 EBEB CA62EA             JZ    :EA62      Then go to Error
172
173                 * Check if next opcode is RST or EI/DI:
174
175 EBEE F3       L3E188  DI
176 EBEF 225100             SHLD  :0051      Save addr current instr
177 EBF2 114C00             LXI   D,:004C
178 EBF5 EB                 XCHG
179 EBF6 225D00             SHLD  :005D      Set addr next instr = 004C
180 EBF9 EB                 XCHG
181 EBFA 7E                 MOV   A,M        Get opcode of instr
182 EBFB E6C7               ANI   :C7
183 EBFD FEC7               CPI   :C7
184 EBFF CA33EC             JZ    :EC33      Jump if RST
185 EC02 3A5F00             LDA   :005F      Get int. mask
186 EC05 CD3EEF             CALL  :EF3E      Store it in TIC; set A=0
                            STA   :FFF9      Reset timer 1 (trap
```

```
188                                             immediate)
189 EC0B 7E                 MOV   A,M           Get opcode of instr
190 EC0C E6F7               ANI   :F7           ) Check if EI or DI
191 EC0E D6F3               SUI   :F3           )
192 EC10 7E                 MOV   A,M           Get inst code
193 EC11 EB                 XCHG                HL = #004C
194 EC12 36FB               MVI   M,:FB         Load EI in #004C
195 EC14 0E03               MVI   C,:03         3 instr bytes to be loaded
196                                             into #004D-F
197 EC16 CA2CEC             JZ    :EC2C         Jump if instr is EI/DI
198
199                 * Load next instruction into 004D-4F:
200
201 EC19 1A          L3E189 LDAX  D             Get instr code
202 EC1A 23          L3E190 INX   H
203 EC1B 77                 MOV   M,A           Store it in #004D-F
204 EC1C 13                 INX   D
205 EC1D 0D                 DCR   C
206 EC1E C219EC             JNZ   :EC19         Next byte of instr
207                 *
208                 * Run LOOK: Register contents:
209                 *
210                 *   If 'normal instr': 004C:    EI.
211                 *                      004D-4F: Next instruction.
212                 *
213                 *   If RST *:          004C-4D: RST */ data **.
214                 *                      004E:    RST 0.
215                 *
216                 *   If EI/DI:          004C:    EI.
217                 *                      004D:    NOP.
218                 *                      004E-50: Next instruction.
219                 *                         This may cause problems,
220                 *                         because 0050 is LOOK init
221                 *                         flag.
222                 *
223 EC21 CD30EF             CALL  :EF30         XRA A; LXI H,#0050
224 EC24 BE                 CMP   M             Check look flag
225 EC25 77                 MOV   M,A           Reset flag
226 EC26 CA63EC             JZ    :EC63         If not Look init: restore
227                                             CPU reg; cont on (005D/E)=
228                                             004C
229 EC29 C39CED             JMP   :ED9C         Else: Restore TIC/GICC/
230                                             CPU reg/int.mask  and 'GO'
231                                             to (005D/E)=004C
232
233                 * If instruction is EI or DI:
234
235 EC2C 324700        L3E191 STA   :0047       Store EI/DI instr
236 EC2F AF                 XRA   A
237 EC30 C31AEC             JMP   :EC1A         Load 00 in #004D, next instr
238                                             in 004E-0050
239
240                 * If a RST instruction:
241
242 EC33 EB          L3E192 XCHG                HL=004C
243 EC34 0E02               MVI   C,:02         Only 2-byte instruction
244 EC36 3EC7               MVI   A,:C7
245 EC38 324E00             STA   :004E         Set (004E)=#C7 (=RST0)
246 EC3B C344EF             JMP   :EF44         RST*/data ** in 004C/D
247
248                 * If PC points to 004D-004E:
249
```

```
250 EC3E 19          L3E193  DAD   D          Set HL=0000-0001. This may
251                                           cause re-entry problems due
252                                           to stack manipulation
253 EC3F 225D00              SHLD  :005D      Save addr next instr (pnts
254                                           to RST0)
255 EC42 C3B0EB              JMP   :EBB0
256                  *
257                  *
258                  *
259 EC45                     END
```

****
* S Y M B O L T A B L E *
****

```
L3E178 EB41   L3E179 EB56   L3E180 EB5D   L3E181 EB6C
L3E182 EB98   L3E183 EBAC   L3E184 EBAE   L3E185 EBB0
L3E186 EBDC   L3E187 EBE2   L3E188 EBEE   L3E189 EC19
L3E190 EC1A   L3E191 EC2C   L3E192 EC33   L3E193 EC3E
LOOKK  EB26
```

```
002                             ORG   :EC45
003                     *
004                     *
005                     *
006                     ************************
007                     * COMPARE DE WITH HL   *
008                     ************************
009                     *
010                     * Exit: AF corrupted, BCDEHL preserved.
011                     *       HL > DE : A=-1, CY=1, Z=0, S=1.
012                     *       HL = DE : A= 0, CY=0, Z=1, S=0.
013                     *       HL < DE : A= 1, CY=0, Z=0, S=0.
014                     *
015 EC45 7B             L3E194  MOV   A,E
016 EC46 95                     SUB   L
017 EC47 7A                     MOV   A,D
018 EC48 9C                     SBB   H
019 EC49 9F                     SBB   A
020 EC4A F8                     RM                  If HL>DE: A=1, S=1
021 EC4B 7B                     MOV   A,E
022 EC4C AD                     XRA   L
023 EC4D D5                     PUSH  D
024 EC4E 5F                     MOV   E,A
025 EC4F 7A                     MOV   A,D
026 EC50 AC                     XRA   H
027 EC51 B3                     ORA   E
028 EC52 D1                     POP   D
029 EC53 37                     STC
030 EC54 C8                     RZ                  If HL=DE: A=0, Z=CY=1
031 EC55 AF                     XRA   A
032 EC56 3C                     INR   A
033 EC57 C9                     RET                 If HL<DE: A=1, CY=0
034                     *
035                     ************************************
036                     * DECREMENT HL, COMPARE HL WITH DE *
037                     ************************************
038                     *
039                     * HL = HL - 1.
040                     *
041                     * Exit: AF corrupted, BCDEHL preserved.
042                     *       HL was 0: CY=1, Z=1.
043                     *       Else:     New HL > DE : CY=0.
044                     *                 New HL = DE :  Z=1.
045                     *                 New HL < DE : CY=1.
046                     *
047 EC58 2B             L3E195  DCX   H
048 EC59 7C                     MOV   A,H
049 EC5A A5                     ANA   L
050 EC5B C601                   ADI   :01
051 EC5D D8                     RC                Ready if old HL was 0
052 EC5E 7D                     MOV   A,L         )
053 EC5F 93                     SUB   E           ) Compare HL - DE
054 EC60 7C                     MOV   A,H         )
055 EC61 9A                     SBB   D           )
056 EC62 C9                     RET
057                     *
058                     ************************
059                     * RESTORE CPU REGISTERS *
060                     ************************
061                     *
062                     * Restores CPU registers AFBCDEHL and PC.
063                     * Continues at PC address.
```

```
064                     * Stackpointer, TICC and GIC are not restored !
065                     *
066 EC63 CDE7ED         L3E196  CALL   :EDE7      Exchange bytes in reg.
067                                               save area
068 EC66 2A5300         L3E197  LHLD   :0053      Get stored PSW
069 EC69 E5                     PUSH   H
070 EC6A F1                     POP    PSW        Restore PSW
071 EC6B 2A5500                 LHLD   :0055
072 EC6E 44                     MOV    B,H
073 EC6F 4D                     MOV    C,L        Restore BC
074 EC70 2A5700                 LHLD   :0057
075 EC73 EB                     XCHG              Restore DE
076 EC74 2A5D00                 LHLD   :005D
077 EC77 E5                     PUSH   H          Addr next instr on stack
078 EC78 2A5900                 LHLD   :0059      Restore HL
079 EC7B C9                     RET               Goto addr in (005D/E)
080                     *
081                     **********************
082                     * CALCULATE DE - HL *
083                     **********************
084                     *
085                     * Exit: HL = DE - HL.
086                     *       BCDE preserved, AF corrupted.
087                     *
088 EC7C 7B             L3E198  MOV    A,E
089 EC7D 95                     SUB    L
090 EC7E 6F                     MOV    L,A        L=E-L
091 EC7F 7A                     MOV    A,D
092 EC80 9C                     SBB    H
093 EC81 67                     MOV    H,A        H=D-H-CY
094 EC82 C9                     RET
095                     *
096                     ************
097                     * M - MOVE *
098                     ************
099                     *
100                     * Moves a block of data (laddr - haddr) given to
101                     * a given destination address (daddr).
102                     *
103                     * Exit: BC: 1st unused destination address.
104                     *       DE: Last source address for direction of
105                     *           movement.
106                     *       HL: 1st unused source address.
107                     *       AF: Corrupted.
108                     *
109 EC83 0E03           MOVEK   MVI    C,:03      Nr of addr allowed
110 EC85 CDDEEA                 CALL   :EADE      Get addr on stack
111 EC88 0D                     DCR    C          3 addr given ?
112 EC89 F262EA                 JP     :EA62      Error if not
113 EC8C C1                     POP    B          daddr in BC
114 EC8D D1                     POP    D          haddr in DE
115 EC8E E1                     POP    H          laddr in HL
116 EC8F E5                     PUSH   H          Save laddr on stack
117 EC90 CD7CEC                 CALL   :EC7C      Calc length of block to
118                                               be moved
119 EC93 DA62EA                 JC     :EA62      Error if wrong inputs
120 EC96 E3                     XTHL              laddr in HL, length on stack
121 EC97 79                     MOV    A,C        )
122 EC98 95                     SUB    L          ) Check if daddr < laddr
123 EC99 78                     MOV    A,B        )
124 EC9A 9C                     SBB    H          )
125 EC9B DAAEEC                 JC     :ECAE      Then jump
```

```
126
127                     * If daddr > laddr:
128
129 EC9E E3                      XTHL            length in HL, laddr on stack
130 EC9F 09                      DAD   B         daddr of highest byte
131 ECA0 44                      MOV   B,H       ) Store it in BC
132 ECA1 4D                      MOV   C,L       )
133 ECA2 E1                      POP   H         laddr in HL
134 ECA3 EB                      XCHG            DE: laddr, HL: haddr
135 ECA4 7E             L3E200   MOV   A,M       Get byte to be moved
136 ECA5 02                      STAX  B         Move it
137 ECA6 0B                      DCX   B         Decr pntr daddr
138 ECA7 CD58EC                  CALL  :EC58     Check if ready
139 ECAA D8                      RC              Then quit
140 ECAB C3A4EC                  JMP   :ECA4     Next byte
141
142                     * If daddr < laddr:
143
144 ECAE E3             L3E201   XTHL            Length in HL, laddr on stack
145 ECAF E1                      POP   H         laddr in HL
146 ECB0 7E             L3E202   MOV   A,M       Get byte
147 ECB1 02                      STAX  B         And move it
148 ECB2 03                      INX   B         Incr pntr daddr
149 ECB3 CD80ED                  CALL  :ED80     INX H; check if ready
150 ECB6 D8                      RC              Then quit
151 ECB7 C3B0EC                  JMP   :ECB0     Next byte
152                     *
153                     *************
154                     * Z - RESET *
155                     *************
156                     *
157                     * Z1: Reset CPU save area 0051-005E. Initialise
158                     *     stackpointer to F900 and save it in 005B/C.
159                     *
160                     * Z2: Sets: current interrupt mask (005F) = #C5,
161                     *           TICC control word (0060) = #FC,
162                     *           GIC  control word (0061) = #1B.
163                     *     Initialises interrupt vector area #0062-#0071.
164                     *     Sets interrupt vector RST 0 to #EB5D.
165                     *
166                     * Z3: Z1 + Z2.
167                     *
168                     * Exit: All registers corrupted.
169                     *
170 ECBA 0E01           ZEROK    MVI   C,:01     Nr of databytes allowed
171 ECBC CDDEEA                  CALL  :EADE     Get hexnr and store it
172                                             on stack
173 ECBF E1                      POP   H         Get hexnr from stack
174 ECC0 7D                      MOV   A,L       into A
175 ECC1 F5                      PUSH  PSW       Save it again
176 ECC2 E602                    ANI   :02
177 ECC4 CAE9EC                  JZ    :ECE9     Jump if Z1 only
178
179                     * If Z2 or Z3:
180
181 ECC7 3EC5                    MVI   A,:C5     Set interrupt mask for
182                                             clock, keyb, ext, timer 1
183 ECC9 325F00                  STA   :005F     Preserve int.mask
184 ECCC 3EF4                    MVI   A,:F4
185 ECCE 00                      NOP
186 ECCF 00                      NOP
187 ECD0 00                      NOP
```

```
188 ECD1 F608                ORI   :08
189 ECD3 216000              LXI   H,:0060
190 ECD6 77                  MOV   M,A        Set TICC contr.word = FC
191 ECD7 23                  INX   H
192 ECD8 361B                MVI   M,:1B      Set GIC contr.word = 1B
193 ECDA CD1ED9              CALL  :D91E      Init int.vector addr
194 ECDD 215DEB              LXI   H,:EB5D
195 ECE0 226200              SHLD  :0062      Set RST0 vector = EB5D
196 ECE3 00                  NOP
197 ECE4 00                  NOP
198 ECE5 00                  NOP
199 ECE6 00                  NOP
200 ECE7 00                  NOP
201 ECE8 00          L3E204  NOP
202 ECE9 F1          ZER01   POP   PSW        Get nr Z instr back
203 ECEA 1F                  RAR              Check for Z2 only
204 ECEB D0                  RNC              Ready if Z2 only
205
206                  * If Z1 or Z3:
207
208 ECEC 0E00                MVI   C,:00      )
209 ECEE 115E00              LXI   D,:005E    ) Init reg. save area:
210 ECF1 215100              LXI   H,:0051    )
211 ECF4 CD54ED              CALL  :ED54      ) Fill (0051-005E) with 00
212 ECF7 2100F9              LXI   H,:F900
213 ECFA F9                  SPHL             Set SP=F900
214 ECFB 225B00              SHLD  :005B      Save SP
215 ECFE C342EA              JMP   :EA42      Return for new inputs
216                  *
217                  ****************
218                  * PRINT SPACE *
219                  ****************
220                  *
221                  * Exit: AFC corrupted, BDEHL preserved.
222                  *
223                  TSP
224 ED01 0E20        LSP     MVI   C,:20
225 ED03 C3B4EE              JMP   :EEB4      Print space
226                  *
227                  ************************************
228                  * SCAN KEYBOARD, PRINT CHARACTER *
229                  ************************************
230                  *
231                  * Gets a keyboard input. Zeroes are ignored.
232                  * The character is printed (if not 'ESC').
233                  *
234                  * Exit: A : Character typed in.
235                  *       Other registers preserved.
236                  *
237 ED06 CDB8EE      CIE     CALL  :EEB8      Scan keyboard
238 ED09 E67F                ANI   :7F        Skip bit 7
239 ED0B CA06ED              JZ    :ED06      Ignore zeroes (await an
240                                           input)
241 ED0E FE12                CPI   :12        'ESC' ?
242 ED10 C5                  PUSH  B
243 ED11 4F                  MOV   C,A        Char in C
244 ED12 C4B4EE              CNZ   :EEB4      Print char if not 'ESC'
245 ED15 79                  MOV   A,C        Char in A
246 ED16 C1                  POP   B
247 ED17 C9                  RET
248                  *
```

```
                        ****************************
                        * PRINT ADDRESS IN ASCII *
                        ****************************
                        *
                        * LADDR: An address in HL is converted to ASCII and
                        *        printed (4 nibbles).
                        * LBYTE: A 1-byte value is printed in ASCII.
                        *
                        * Entry: LADDR: Address in HL.
                        *        LBYTE: Value in A.
                        * Exit:  AFC corrupted, BDEHL preserved.
                        *
ED18 7C         LADDR   MOV   A,H       Hibyte in A.
ED19 CD1DED             CALL  :ED1D     Print both nibbles in ASCII
ED1C 7D                 MOV   A,L       Lobyte in A
ED1D F5         LBYTE   PUSH  PSW       Preserve hex value 2 char
ED1E 07                 RLC             )
ED1F 07                 RLC             ) Move hinibble
ED20 07                 RLC             ) into lonibble
ED21 07                 RLC             )
ED22 CD26ED             CALL  :ED26     Print lonibble
ED25 F1                 POP   PSW       Restore byte
ED26 E60F       L3E210  ANI   :0F       Lonibble only
ED28 CD40ED             CALL  :ED40     Convert it to ASCII
ED2B 4F                 MOV   C,A       And store it in C
ED2C C3B4EE             JMP   :EEB4     Print char in C
                        *
                        ****************
                        * PRINT STRING *
                        ****************
                        *
                        * Entry: HL points to string. End of string is 00.
                        * Exit:  HL points to 00 at end of string.
                        *        AFC corrupted, BDE preserved.
                        *
ED2F 7E         L3E211  MOV   A,M       Get byte from string
ED30 4F                 MOV   C,A       into C
ED31 B7                 ORA   A         Byte = 00 ?
ED32 C8                 RZ              Then ready
ED33 CDB4EE             CALL  :EEB4     Print char in C
ED36 23                 INX   H         Pnts to next char
ED37 C32FED             JMP   :ED2F     Print next char
                        *
                        ************************
                        * PRINT CARRIAGE RETURN *
                        ************************
                        *
                        * Exit: AFC corrupted, BDEHL preserved.
                        *
ED3A 0E0D       LCRLF   MVI   C,:0D
ED3C CDB4EE             CALL  :EEB4     Print 'CR'
ED3F C9                 RET
                        *
                        ******************************
                        * CONVERT HEX NIBBLE TO ASCII *
                        ******************************
                        *
                        * Entry: Hex value in A.
                        * Exit:  ASCII value in A.
                        *        BCDEHL preserved. F corrupted.
                        *
ED40 C630       L3E213  ADI   :30       Convert
```

```
312 ED42 FE3A              CPI   :3A       Number 0-9?
313 ED44 D8                RC             Then ready
314 ED45 C607              ADI   :07       Convert A-F
315 ED47 C9                RET
316                  *
317                  ************
318                  * F - FILL *
319                  ************
320                  *
321                  * Fills a memory area between given boundaries
322                  * with given data.
323                  *
324                  * Exit: CY=1. All registers corrupted.
325                  *
326 ED48 0E03        FILLK MVI   C,:03     Nr of addr/data allowed
327 ED4A CDDEEA             CALL  :EADE     Get addr/data on stack
328 ED4D 0D                DCR   C         3 blocks given?
329 ED4E F262EA            JP    :EA62     Error if not
330 ED51 C1                POP   B         Data in C
331 ED52 D1                POP   D         haddr in DE
332 ED53 E1                POP   H         laddr in HL
333 ED54 71          FILL  MOV   M,C       Data into memory
334 ED55 CD80ED            CALL  :ED80     INX H; check if ready
335 ED58 D8                RC             Then quit
336 ED59 C354ED            JMP   :ED54     Fill next addr
337                  *
338                  ******************
339                  * S - SUBSTITUTE *
340                  ******************
341                  *
342 ED5C 0E01        SUBSK MVI   C,:01     Nr of addr allowed
343 ED5E CDDEEA            CALL  :EADE     Get addr on stack
344 ED61 E1                POP   H         Addr in HL
345 ED62 7E                MOV   A,M       Get contents of addr
346 ED63 CD1DED            CALL  :ED1D     Print it in ASCII
347 ED66 AF                XRA   A
348 ED67 CD6AEE            CALL  :EE6A     Evt. modify contents
349 ED6A D262ED            JNC   :ED62     Next addr
350 ED6D C9                RET
351                  *
352                  ***************
353                  * X - EXAMINE *
354                  ***************
355                  *
356 ED6E 219DEE      EXAMK LXI   H,:EE9D   Startaddr register table
357 ED71 115300            LXI   D,:0053   Startaddr CPU save area
358 ED74 C339EE            JMP   :EE39     Go to display routine
359                  *
360                  ************************
361                  * V - VECTOR EXAMINE *
362                  ************************
363                  *
364 ED77 21A8EE      VECXK LXI   H,:EEA8   Startaddr vector table
365 ED7A 115F00            LXI   D,:005F   Startaddr vector area
366 ED7D C339EE            JMP   :EE39     Go to display routine
367                  *
368                  ************************************
369                  * INCREMENT HL AND COMPARE WITH DE *
370                  ************************************
371                  *
372                  * Exit: HL was FFFF: CY=1, Z=1.
373                  *       Else: New HL < DE : CY=0.
```

```
374                     *               New HL = DE :  Z=1.
375                     *               New HL > DE : CY=1.
376                     *       BCDE preserved, HL=HL+1, AF corrupted.
377                     *
378 ED80 23             INXCK   INX   H
379 ED81 37                     STC                   CY=1
380 ED82 7C                     MOV   A,H
381 ED83 B5                     ORA   L
382 ED84 C8                     RZ                    Abort if new HL is 0000
383 ED85 7B                     MOV   A,E
384 ED86 95                     SUB   L
385 ED87 7A                     MOV   A,D
386 ED88 9C                     SBB   H
387 ED89 C9                     RET
388                     *
389                     **********
390                     * G - GO *
391                     **********
392                     *
393                     * Reads one field if given.
394                     * If no field given: Restores CPU registers, goes
395                     *    to PC address. Returnaddress is #EA42 (into
396                     *    command loop).
397                     * If field given: Saves it as PC, initialises TICC
398                     *    and GIC. Returnaddress is #EA04. Goes to the
399                     *    given address.
400                     * REMARK: The stackpointer is never restored !
401                     *
402 ED8A 0E01           GOK     MVI   C,:01
403 ED8C CD07EB                 CALL  :EB07           Scan keyb; addr on stack
404 ED8F 0D                     DCR   C               No addr given?
405 ED90 F5                     PUSH  PSW
406 ED91 CC56EB                 CZ    :EB56           Then check if 'CR'
407 ED94 F1                     POP   PSW
408 ED95 CA63EC                 JZ    :EC63           No addr: restore CPU reg
409                                                   (but not SP/TICC/GIC !) and
410                                                   go to addr in 005D/E
411 ED98 E1                     POP   H               'GO' addr in HL
412 ED99 225D00                 SHLD  :005D           Save it
413 ED9C 2A6000         L3E221  LHLD  :0060           Get GIC/TICC init values
414 ED9F 7C                     MOV   A,H             GIC init value in A
415 EDA0 CDD5ED                 CALL  :EDD5           Init GIC
416 EDA3 7D                     MOV   A,L             TICC init value in A
417 EDA4 CDB7ED                 CALL  :EDB7           Init TICC
418 EDA7 3A5F00                 LDA   :005F           Get current int mask
419 EDAA 32F8FF                 STA   :FFF8           Set int mask
420 EDAD CDE7ED                 CALL  :EDE7           exch. bytes in 0051-5E
421 EDB0 2104EA                 LXI   H,:EA04         Returnaddr for 'GO'
422 EDB3 E5                     PUSH  H               Save EA04 on stack
423 EDB4 C366EC                 JMP   :EC66           Restore CPU reg and 'GO'
424                     *
425                     * INITIALISE TICC:
426                     *
427                     * Entry: A: Initial value TICC command word (#FC).
428                     * Exit:  AFBC corrupted, DEHL preserved.
429                     *
430 EDB7 47             L3E222  MOV   B,A             Init.value in B
431 EDB8 07                     RLC                   A=F9
432 EDB9 F5                     PUSH  PSW             On stack: A=F9, CY=1
433 EDBA 07                     RLC                   A=F3
434 EDBB 07                     RLC                   A=E7
435 EDBC 07                     RLC                   A=CF
```

```
436 EDBD E607                ANI   :07       A=07
437 EDBF 4F                  MOV   C,A       C=07
438 EDC0 AF                  XRA   A         A=0
439 EDC1 37                  STC             CY=1
440 EDC2 17          L3E223  RAL             ) On exit: A=80,
441 EDC3 0D                  DCR   C         ) C=FF, CY=0
442 EDC4 F2C2ED              JP    :EDC2     )
443 EDC7 4F                  MOV   C,A       C=80
444 EDC8 F1                  POP   PSW       A=F9, CY=1
445 EDC9 79                  MOV   A,C       A=80
446 EDCA 1F                  RAR             A=C0
447 EDCB 32F5FF              STA   :FFF5     Set comm.rate reg for 9600
448                                          baud, 1 stop bit
449 EDCE 78                  MOV   A,B       A=FC
450 EDCF E60F                ANI   :0F       A=0C
451 EDD1 32F4FF              STA   :FFF4     Set cmd reg for IN7, INTA
452                                          enable
453 EDD4 C9                  RET
454                  *
455                  * INITIALISE GIC:
456                  *
457                  * This initialisation cancels the initial setting
458                  * done during 'power-on' by 3EF90. Only used during
459                  * 'GO'.
460                  * There seems to be some bug in the routine. All
461                  * ports are set to input, and then data is written
462                  * into one of this ports (PB - FE01). The function
463                  * of L3E225 is nonsense (?!).
464                  *
465                  * Entry: A: Initial value GIC command word (#1B).
466                  * Exit:  AFBC corrupted, DEHL preserved.
467                  *
468 EDD5 F5          L3E224  PUSH  PSW       Init value on stack, CY=0
469 EDD6 0103FE              LXI   B,:FE03   Addr command word
470 EDD9 F680                ORI   :80       A=9B
471 EDDB 02                  STAX  B         Set all ports to input
472 EDDC 0D                  DCR   C         BC=FE02
473 EDDD F1                  POP   PSW       A=1B, CY=0
474 EDDE 07                  RLC             A=36, CY=0
475 EDDF 9F                  SBB   A         A=0
476 EDE0 0B          L3E225  DCX   B         BC=FE01
477 EDE1 02                  STAX  B         (FE01)=00 (?!)
478 EDE2 0D                  DCR   C         BC=FE00
479 EDE3 F2E0ED              JP    :EDE0     Writes 00 in non-existing
480                                          address FDFE (?!)
481 EDE6 C9                  RET
482                  *
483                  ******************************************
484                  * EXCHANGE BYTES IN REGISTER SAVE AREA *
485                  ******************************************
486                  *
487                  * The hibytes and the lobytes of the addresses
488                  * 0053/0054 thru 0059/5A are exchanged which
489                  * each other.
490                  *
491                  * Exit: AFBCHL corrupted. DE preserved.
492                  *
493 EDE7 215300      L3E238  LXI   H,:0053   Startaddr
494 EDEA 3E04                MVI   A,:04     Nr of addr to be exchanged
495 EDEC 46          L3E226  MOV   B,M       1st byte in B
496 EDED 23                  INX   H         Pnts to next location
497 EDEE 4E                  MOV   C,M       2nd byte in C
```

```
498 EDEF 70                    MOV   M,B       1st byte in 2nd location
499 EDF0 2B                    DCX   H
500 EDF1 71                    MOV   M,C       2nd byte in 1st location
501 EDF2 23                    INX   H
502 EDF3 23                    INX   H         Points to next addr
503 EDF4 3D                    DCR   A         Update counter
504 EDF5 C2ECED                JNZ   :EDEC     Next addr if not ready
505 EDF8 C9                    RET
506                  *
507                  *
508                  *
509 EDF9                       END
```

****
* SYMBOL TABLE *
****

CIE	ED06	EXAMK	ED6E	FILL	ED54	FILLK	ED48
GOK	ED8A	INXCK	ED80	L3E194	EC45	L3E195	EC58
L3E196	EC63	L3E197	EC66	L3E198	EC7C	L3E200	ECA4
L3E201	ECAE	L3E202	ECB0	L3E204	ECE8	L3E210	ED26
L3E211	ED2F	L3E213	ED40	L3E221	ED9C	L3E222	EDB7
L3E223	EDC2	L3E224	EDD5	L3E225	EDE0	L3E226	EDEC
L3E238	EDE7	LADDR	ED18	LBYTE	ED1D	LCRLF	ED3A
LSP	ED01	MOVEK	EC83	SUBSK	ED5C	TSP	ED01
VECXK	ED77	ZERO1	ECE9	ZEROK	ECBA		

```
002                             ORG   :EDF9
003                     *
004                     *
005                     *
006                     ********************************
007                     * PRINT CONTENTS CPU SAVE AREA *
008                     ********************************
009                     *
010                     * Entry: HL: Points to register save area.
011                     *        msb A = 0: 1 byte to be printed.
012                     *        msb A = 1: 2 bytes to be printed.
013                     * Exit:  msb A = 0: AFC corrupted, BDEHL preserved.
014                     *        msb A = 1: AFCDE corrupted, BHL preserved.
015                     *
016 EDF9 B7             L3E227  ORA   A         Test flags
017 EDFA 7E                     MOV   A,M       Get contents save area
018 EDFB F21DED                 JP    :ED1D     1 byte: print it in ASCII
019
020                     * If 2 bytes:
021
022 EDFE 5E                     MOV   E,M       Get contents in E
023 EDFF 23                     INX   H         Next addr
024 EE00 56                     MOV   D,M       Its contents in D
025 EE01 2B                     DCX   H         Restore HL
026 EE02 E5                     PUSH  H         Save it on stack
027 EE03 EB                     XCHG            Contents addr in HL
028 EE04 CD18ED                 CALL  :ED18     Print 2 bytes in ASCII
029 EE07 E1                     POP   H         Restore HL
030 EE08 C9                     RET
031                     *
032                     ****************************************
033                     * V+X: PRINT ROUTINE IF REGISTER GIVEN *
034                     ****************************************
035                     *
036                     * Entry: HL: Startaddress symbol table.
037                     *        DE: Startaddress CPU save area
038                     *        A : Last input character.
039                     * Exit:  All registers corrupted.
040                     *
041 EE09 47             L3E228  MOV   B,A       Last input in B
042 EE0A CD01ED                 CALL  :ED01     Print space
043 EE0D 7E             L3E229  MOV   A,M       Get symbol from table
044 EE0E E67F                   ANI   :7F       Skip bit 7
045 EE10 CA62EA                 JZ    :EA62     Error if symbol=0
046 EE13 B8                     CMP   B         Compare input with symbol
047 EE14 CA22EE                 JZ    :EE22     Jump if identical
048 EE17 7E                     MOV   A,M       Get symbol
049 EE18 07                     RLC             Check for msb set
050 EE19 23                     INX   H         Next symbol
051 EE1A 13                     INX   D         Next memory area
052 EE1B D20DEE                 JNC   :EE0D     Try next symbol
053 EE1E 13                     INX   D         Next mem. area for '2 byte'
054                                             symbols
055 EE1F C30DEE                 JMP   :EE0D     Try again
056
057                     * If symbol found:
058
059 EE22 D5             L3E230  PUSH  D         Save addr in mem.area
060 EE23 7E             L3E231  MOV   A,M       Get symbol
061 EE24 E3                     XTHL            HL: addr mem.area;
062                                             stack: addr symbol
063 EE25 F5                     PUSH  PSW       Save symbol
```

```
064 EE26 CDF9ED             CALL  :EDF9     Print contents mem.area
065 EE29 F1                 POP   PSW       Retrieve symbol
066 EE2A CD6AEE             CALL  :EE6A     Exchange mem.contents,
067                                         go to next one
068 EE2D DA37EE             JC    :EE37     Jump if 'ESC'
069 EE30 E3                 XTHL            HL: addr symbol,
070                                         stack: next mem.area
071 EE31 23                 INX   H         addr next symbol
072 EE32 7E                 MOV   A,M       Get symbol
073 EE33 B7                 ORA   A         Set flags
074 EE34 C223EE             JNZ   :EE23     Not all symbols done
075                 *
076 EE37 D1         L3E232  POP   D         Retrieve next mem.area
077 EE38 C9                 RET
078                 *
079                 ************************
080                 * V+X: DISPLAY ROUTINE *
081                 ************************
082                 *
083                 * Displays registers at succesive memory locations.
084                 *
085                 * Entry: DE: startaddress memory area to be
086                 *            displayed.
087                 *         HL: Startaddress symbol table.
088                 * Exit:  All registers corrupted.
089                 *
090 EE39 CD06ED     L3E233  CALL  :ED06     Scan keyb, print char
091 EE3C FE0D             CPI   :0D       'CR' ?
092 EE3E C209EE             JNZ   :EE09     Jump if also byte given
093
094                 * If only 'V' or 'X':
095
096 EE41 00                 NOP
097 EE42 00                 NOP
098 EE43 00                 NOP
099 EE44 D5         L3E234  PUSH  D         Save startaddr mem area
100 EE45 4E                 MOV   C,M       Get symbol in C
101 EE46 CDB4EE             CALL  :EEB4     Print symbol
102 EE49 0E3D               MVI   C,:3D
103 EE4B CDB4EE             CALL  :EEB4     Print '='
104 EE4E 7E                 MOV   A,M       Get symbol in A
105 EE4F B7                 ORA   A         Check flags
106 EE50 E3                 XTHL            HL: startaddr mem.area
107                                         stack: startaddr symboltable
108 EE51 F5                 PUSH  PSW       Save symbol + flags
109 EE52 CDF9ED             CALL  :EDF9     Print contents mem.area
110 EE55 F1                 POP   PSW       Retrieve symbol and flags
111 EE56 23                 INX   H
112 EE57 F25BEE             JP    :EE5B     Jump if '1 byte' symbol
113
114                 * If 2-byte symbol:
115
116 EE5A 23                 INX   H
117 EE5B 00         L3E235  NOP
118 EE5C 00                 NOP
119 EE5D 00                 NOP
120 EE5E E3                 XTHL            HL: addr symbol
121                                         stack: addr next mem.area
122 EE5F 23                 INX   H         Next symbol
123 EE60 D1                 POP   D         Get addr next mem.area
124 EE61 7E                 MOV   A,M       Get symbol
125 EE62 B7                 ORA   A
```

```
126 EE63 C8              RZ              Quit if ready
127 EE64 CD01ED          CALL  :ED01     Print space
128 EE67 C344EE          JMP   :EE44     Next one
129                      *
130                      ******************************
131                      * V+X: EVT. MODIFY CONTENTS *
132                      ******************************
133                      *
134                      * Entry: HL: Memory address.
135                      *         A : Symbol.
136                      *
137 EE6A B7              L3E326  ORA   A         Set flags on symbol
138 EE6B F5                      PUSH  PSW       and save it
139 EE6C 0E2D                    MVI   C,:2D
140 EE6E CDB4EE                  CALL  :EEB4     Print '-'
141 EE71 E5                      PUSH  H         Save addr mem.area
142 EE72 0E01                    MVI   C,:01     Nr of datablocks allowed
143 EE74 CD07EB                  CALL  :EB07     Get input on stack; last
144                                              byte typed in in B
145 EE77 0D                      DCR   C
146 EE78 CA87EE                  JZ    :EE87     Jump if incorrect input
147 EE7B D1                      POP   D         Data typed in in DE
148 EE7C E1                      POP   H         Get addr mem.area
149 EE7D F1                      POP   PSW       Get symbol and flags
150 EE7E 73                      MOV   M,E       Change memory contents
151 EE7F F28DEE                  JP    :EE8D     Jump if '1 byte' symbol
152
153                      * If 2-byte symbol:
154
155 EE82 23                      INX   H
156 EE83 72                      MOV   M,D       Change 2nd byte
157 EE84 C38DEE                  JMP   :EE8D
158                      *
159 EE87 E1              L3E327  POP   H         Retrieve addr mem.area
160 EE88 F1                      POP   PSW       Retrieve symbol + flags
161 EE89 F28DEE                  JP    :EE8D     If '1 byte' symbol
162 EE8C 23                      INX   H         Add. INX H if 2-byte symbol
163 EE8D 23              L3E328  INX   H         Next mem.area
164 EE8E 78                      MOV   A,B       Get last input
165 EE8F FE0D                    CPI   :0D
166 EE91 C8                      RZ              Ready if 'CR'
167 EE92 FE20                    CPI   :20
168 EE94 C8                      RZ              Ready if 'SP'
169 EE95 FE12                    CPI   :12
170 EE97 37                      STC             Abort with CY=1 if 'ESC'
171 EE98 C8                      RZ
172 EE99 C362EA                  JMP   :EA62     Else: wrong input: Error
173                      *
174                      ******************************
175                      * SYMBOL TABLE EXAMINE (X) *
176                      ******************************
177                      *
178                      * The msb is '1' for symbols of two-byte
179                      * registers.
180                      *
181 EE9C C9              L3E405  DATA  :C9       I (addr current instr)
182 EE9D 41                      DATA  :41       A
183 EE9E 46                      DATA  :46       F (flags)
184 EE9F 42                      DATA  :42       B
185 EEA0 43                      DATA  :43       C
186 EEA1 44                      DATA  :44       D
187 EEA2 45                      DATA  :45       E
```

```
188 EEA3 48                 DATA  :48       H
189 EEA4 4C                 DATA  :4C       L
190 EEA5 D3                 DATA  :D3       S (stackpointer)
191 EEA6 D0                 DATA  :D0       P (program counter)
192 EEA7 00                 DATA  :00
193                 *
194                 ********************************
195                 * SYMBOL TABLE VECTOR EXAMINE *
196                 ********************************
197                 *
198                 * The msb is '1' for symbols of two-byte
199                 * registers.
200                 *
201 EEA8 4D          L3E406  DATA  :4D       M (TICC int mask)
202 EEA9 54                  DATA  :54       T (TICC cmd + comm.reg)
203 EEAA 47                  DATA  :47       G (GIC cmd word)
204 EEAB B0                  DATA  :B0       0  )
205 EEAC B1                  DATA  :B1       1  )
206 EEAD B2                  DATA  :B2       2  )
207 EEAE B3                  DATA  :B3       3  ) Interrupt vectors
208 EEAF B4                  DATA  :B4       4  ) addresses
209 EEB0 B5                  DATA  :B5       5  )
210 EEB1 B6                  DATA  :B6       6  )
211 EEB2 B7                  DATA  :B7       7  )
212 EEB3 00                  DATA  :00
213                 *
214                 ********************
215                 * PRINT CHARACTER *
216                 ********************
217                 *
218                 * Entry: C: Character to be printed.
219                 * Exit:  AF corrupted, BCDEHL preserved.
220                 *
221 EEB4 79          C0      MOV   A,C       Char in A
222 EEB5 C360DD              JMP   :DD60     Print char
223                 *
224                 ********************************
225                 * SCAN KEYBOARD; IGNORE BREAK *
226                 ********************************
227                 *
228                 * Exit: Character in A.
229                 *       BCDEHL preserved.
230                 *
231 EEB8 CD83EF      C1      CALL  :EF83     Scan keyboard
232 EEBB DAB8EE              JC    :EEB8     Ignore break
233 EEBE CAB8EE              JZ    :EEB8     Wait for input
234 EEC1 C9                  RET
235                 *
236                 ************************************
237                 * SCAN KEYBOARD FOR BREAK PRESSED *
238                 ************************************
239                 *
240                 * Exit: CY=0: No break.
241                 *       CY=1: Break pressed.
242                 *       A corrupted, BCDEHL preserved.
243                 *
244 EEC2 CDA5D6      BREAK   CALL  :D6A5     Scan keyb for new inputs
245 EEC5 D0                  RNC             abort if Break not pressed
246 EEC6 C36AEA              JMP   :EA6A     Restore SP, wait for
247                                           new inputs
248                 *
249                 *
```

```
                        **********************
                        * CASSETTE ROUTINES *
                        **********************
                        *
EEC9 C3C502     L3E335  JMP   :02C5     WOPEN
                        *
EECC FF                 DATA  :FF
EECD FF                 DATA  :FF
EECE FF                 DATA  :FF
                        *
EECF C3C802     L3E336  JMP   :02C8     WBLK
                        *
EED2 FF                 DATA  :FF
EED3 FF                 DATA  :FF
EED4 FF                 DATA  :FF
                        *
EED5 C3CB02     L3E337  JMP   :02CB     WCLOSE
                        *
EED8 C3CE02     L3E338  JMP   :02CE     ROPEN
                        *
EEDB FF                 DATA  :FF
EEDC FF                 DATA  :FF
EEDD FF                 DATA  :FF
                        *
EEDE C3D102     L3E339  JMP   :02D1     RBLK
                        *
EEE1 C3D402     L3E340  JMP   :02D4     RCLOSE
                        *
                        *************
                        * W - WRITE *
                        *************
                        *
                        * Requires 2 address fields + evt. name.
                        * Filetype is '1'. Writes startaddress of datablock
                        * + data + trailer on tape.
                        *
EEE4 0E02       L3E341  MVI   C,:02     Nr of addr allowed
EEE6 CDDEEA             CALL  :EADE     Scan keyb, addr on stack
EEE9 0D                 DCR   C         2 addr given ?
EEEA F262EA             JP    :EA62     Error if not
EEED CD48EF             CALL  :EF48     Evt. name in input buffer
EEF0 3E31               MVI   A,:31     File type byte
EEF2 00                 NOP
EEF3 00                 NOP
EEF4 00                 NOP
EEF5 00                 NOP
EEF6 00                 NOP
EEF7 00                 NOP
EEF8 CDC9EE             CALL  :EEC9     Write file header on tape
EEFB D1                 POP   D         Get haddr from stack
EEFC 13                 INX   D         Incr it
EEFD E1                 POP   H         Get laddr from stack
EEFE CD63EF             CALL  :EF63     Write startaddr on tape
EF01 7B                 MOV   A,E       )
EF02 95                 SUB   L         )
EF03 5F                 MOV   E,A       ) Calc length of data
EF04 7A                 MOV   A,D       ) block, result in DE
EF05 9C                 SBB   H         )
EF06 57                 MOV   D,A       )
EF07 00                 NOP
EF08 CDCFEE             CALL  :EECF     Write datablock on tape
EF0B CDD5EE             CALL  :EED5     Write file trailer
```

```
312 EF0E C9                 RET
313                 *
314                 ************
315                 * R - READ *
316                 ************
317                 *
318                 * One address is allowed. An evt. name is stored
319                 * in the EBUF. Reads header, startaddress and data
320                 * from tape. Errorcheck only on data.
321                 *
322                 * Exit: HL: 1st address above file loaded.
323                 *       BC: Evt. offset.
324                 *       AFDE corrupted.
325                 *
326 EF0F 0E01       RHEXK   MVI   C,:01     Nr of addr allowed
327 EF11 CDDEEA             CALL  :EADE     Scan keyb; addr on stack
328 EF14 CD48EF             CALL  :EF48     Evt name in input buffer
329 EF17 0631             MVI   B,:31     File type byte
330 EF19 00              NOP
331 EF1A 00              NOP
332 EF1B 00              NOP
333 EF1C 0E00            MVI   C,:00
334 EF1E CDD8EE          CALL  :EED8     Read file header
335 EF21 1100F9          LXI   D,:F900   Max addr to write data int
336 EF24 C1              POP   B         Get evt offset from stack
337 EF25 CD74EF          CALL  :EF74     Read startaddr from tape
338 EF28 09              DAD   B         Add offset
339 EF29 CD8AEF          CALL  :EF8A     Read data block + trailer
340 EF2C D262EA          JNC   :EA62     Print 'error' if reading
341                                      error
342 EF2F C9              RET
343                 *
344                 ****************************************
345                 * RST 0: POINTER TO 'LOOK'-FLAG IN HL *
346                 ****************************************
347                 *
348                 * Part of 3EC21.
349                 *
350                 * Exit: A=0, BCDE preserved.
351                 *
352 EF30 AF          L3E343  XRA   A
353 EF31 215000           LXI   H,:0050
354 EF34 C9              RET
355                 *
356                 ********************
357                 * INITIALISE RST 0 *
358                 ********************
359                 *
360                 * Entry: On stack: Returnaddress #EA42.
361                 *
362 EF35 3EFB       L3E344  MVI   A,:FB     Instr code for EI in A
363 EF37 324700           STA   :0047     Save it
364 EF3A D1              POP   D         Get EA42 in DE
365 EF3B C3E2EB          JMP   :EBE2     Into RST 0
366                 *
367                 ************************
368                 * SET INTERRUPT MASK *
369                 ************************
370                 *
371                 * Part of RST0 (3EC05).
372                 *
373                 * Entry: A: Value for TICC interrupt mask.
```

```
374                     * Exit:  A=0, F corrupted, BCDEHL preserved.
375                     *
376 EF3E 32F8FF         L3E345  STA   :FFF8     Set TICC int mask
377 EF41 AF                     XRA   A         A=0
378 EF42 C9                     RET
379                     *
380 EF43 FF                     DATA  :FF
381                     *
382                     ****************************
383                     * part of RST 0 (3EC3B) *
384                     ****************************
385                     *
386 EF44 2B             L3E346  DCX   H
387 EF45 C319EC                 JMP   :EC19     Into RST 0
388                     *
389                     ********************************
390                     * W+R: NAME IN INPUT BUFFER *
391                     ********************************
392                     *
393                     * Names >126 characters destroy BASIC pointers.
394                     *
395                     * Entry: Last character typed in, in B.
396                     * Exit:  BCD preserved. HL points to EBUF.
397                     *        A= 0: No file name given.
398                     *        A<>0: File name given.
399                     *
400 EF48 213E01           L3E347  LXI   H,:013E   Startaddr EBUF
401 EF4B 1EFF                     MVI   E,:FF
402 EF4D 78                       MOV   A,B       Last char in A
403 EF4E D60D                     SUI   :0D       'CR' ?
404 EF50 77                       MOV   M,A       (13E) is 0 if CR
405 EF51 C8                       RZ              Quit if no name given
406
407                     * If name given:
408
409 EF52 E5                       PUSH  H         Save startaddr EBUF
410 EF53 2C             L3E356  INR   L         Points to next loc
411 EF54 1C                     INR   E         Calc length
412 EF55 CD06ED                 CALL  :ED06     Scan keyb, print char
413 EF58 77                     MOV   M,A       Char into EBUF
414 EF59 FE0D                   CPI   :0D
415 EF5B C253EF                 JNZ   :EF53     Next char if not 'CR'
416 EF5E E1                     POP   H         Retrieve startaddr EBUF
417 EF5F 73                     MOV   M,E       Store length name in 013E
418 EF60 C9                     RET
419                     *
420                     ***************
421                     * (Not used) *
422                     ***************
423                     *
424 EF61 E1             L3E348  POP   H
425 EF62 C9                     RET
426                     *
427                     ********************************
428                     * WRITE STARTADDRESS ON TAPE *
429                     ********************************
430                     *
431                     * Entry: HL: Startaddress.
432                     * Exit:  AF corrupted, BCDEHL preserved.
433                     *
434 EF63 D5             L3E349  PUSH  D
435 EF64 E5                     PUSH  H
```

```
436 EF65 223E01             SHLD  :013E     Startaddr in EBUF
437 EF68 213E01             LXI   H,:013E   Startaddr to write from
438 EF6B 110200             LXI   D,:0002   Length
439 EF6E CDCFEE             CALL  :EECF     Write addr on tape
440 EF71 E1                 POP   H
441 EF72 D1                 POP   D
442 EF73 C9                 RET
443                 *
444                 ********************************
445                 * READ STARTADDRESS FROM TAPE *
446                 ********************************
447                 *
448                 * Exit: HL: Startaddress.
449                 *       CY=1: No reading error.
450                 *       CY=0: Reading error, errorcode in A.
451                 *       BCDE preserved.
452                 *
453 EF74 D5        L3E350  PUSH  D
454 EF75 213E01             LXI   H,:013E   Addr EBUF
455 EF78 114101             LXI   D,:0141   Addr after addr in EBUF
456 EF7B CDDEEE             CALL  :EEDE     Read block from tape
457 EF7E D1                 POP   D
458 EF7F 2A3E01             LHLD  :013E     Startaddr in HL
459 EF82 C9                 RET
460                 *
461                 ******************
462                 * SCAN KEYBOARD *
463                 ******************
464                 *
465                 * Part of 3EEB8. Scans keyboard. Returns
466                 * any key received.
467                 *
468                 * Exit: A: Key received.
469                 *       BCDEHL preserved.
470                 *       CY=1: Break pressed.
471                 *
472 EF83 AF        L3E351  XRA   A
473 EF84 32B902             STA   :02B9     Enable complete keyb scan
474 EF87 C3BED6             JMP   :D6BE     Scan keyboard
475                 *
476                 ************************
477                 * READ DATA FROM TAPE *
478                 ************************
479                 *
480                 * Part of READ (3EF29).
481                 *
482 EF8A CDDEEE    L3E352  CALL  :EEDE     Read block from tape
483 EF8D C3E1EE             JMP   :EEE1     Stop reading
484                 *
485                 ********************************
486                 * DCE INITIALISATION ROUTINE *
487                 ********************************
488                 *
489                 * Part of RESET (C719). Bootstrap for disc drive.
490                 * Sets GIC in initialisation status. Checks if any
491                 * input is received from the DCE-bus and performs
492                 * the received instructions.
493                 *
494                 * Exit: A=#EE if no DCE-inputs available.
495                 *
496 EF90 3E98      L3E353  MVI   A,:98
497 EF92 3203FE             STA   :FE03     PA+PCH in, PB+PCL out
```

```
498 EF95 3E07              MVI   A,:07
499 EF97 3203FE            STA   :FE03     PC3=1 —>
500 EF9A 3E01              MVI   A,:01
501 EF9C 3201FE            STA   :FE01     Output PB: 01
502 EF9F 3E01              MVI   A,:01
503 EFA1 3203FE            STA   :FE03     PC0=1
504 EFA4 010010            LXI   B,:1000
505 EFA7 3A02FE    L3E355  LDA   :FE02     Get input from PCH
506 EFAA E620              ANI   :20       Bit 5 only
507 EFAC C2B8EF            JNZ   :EFB8     Jump if inputs received
508
509                * If no inputs:
510
511 EFAF 0B                DCX   B         )
512 EFB0 78                MOV   A,B       ) Wait loop until C=#10
513 EFB1 B1                ORA   C         )
514 EFB2 C2A7EF            JNZ   :EFA7     )
515 EFB5 3EEE              MVI   A,:EE     A=EE if no inputs
516 EFB7 C9                RET
517                *
518                * DCE BOOTSTRAP INPUT ROUTINE:
519                *
520                * Loads MLP inputs from the DCE-bus into the
521                * stackbottom and goes to it.
522                *
523 EFB8 1100F8    L3E354  LXI   D,:F800   Addr stackbottom
524 EFBB 3E05      L3E357  MVI   A,:05
525 EFBD 3203FE            STA   :FE03     PC2=1
526 EFC0 3A02FE    L3E358  LDA   :FE02     Get input from PC
527 EFC3 E680              ANI   :80       Bit 7 only
528 EFC5 CAC0EF            JZ    :EFC0     Wait for change to high
529 EFC8 3A00FE            LDA   :FE00     Get input from PA
530 EFCB 12                STAX  D         Save input in stack area
531 EFCC 13                INX   D         Point to next loc
532 EFCD 3E04              MVI   A,:04
533 EFCF 3203FE            STA   :FE03     PC2=0
534 EFD2 3A02FE    L3E359  LDA   :FE02     Get input from PC
535 EFD5 E680              ANI   :80       Bit 7 only
536 EFD7 C2D2EF            JNZ   :EFD2     Wait for change to low
537 EFDA 3A02FE            LDA   :FE02     Get input from PC
538 EFDD E620              ANI   :20       Bit 5 only
539 EFDF C2BBEF            JNZ   :EFBB     Again if high
540 EFE2 3E06              MVI   A,:06
541 EFE4 323EFE            STA   :FE3E     (FE3E hardwarewise read as
542                                        FE02). PC=06
543 EFE7 3A02FE    L3E360  LDA   :FE02     Get input from PC
544 EFEA E620              ANI   :20       Bit 5 only
545 EFEC CAE7EF            JZ    :EFE7     Wait for change to high
546 EFEF C300F8            JMP   :F800     Go to stackbottom
547                *
548 EFF2 FF                DATA  :FF
549 EFF3 FF                DATA  :FF
550                *
551                **********************
552                * SCAN 'DINC' INPUT *
553                **********************
554                *
555                * Part of 3E935. Default 'DINC' is RS232 input.
556                *
557 EFF4 CDB4DD    L3E361  CALL  :DDB4     Get input from DINC
558 EFF7 CA35E9             JZ    :E935     Scan keyb if no DINC input
```

```
560                    * If inputs from DINC:
561
562 EFFA 3E01                  MVI   A,:01
563 EFFC 329602                STA   :0296      Set INSW for DINC input
564 EFFF C9                    RET
565                    *
566                    *
567                    *
568 F000                       END
```

****
* SYMBOL TABLE *
****

BREAK	EEC2	CI	EEB8	CO	EEB4	L3E227	EDF9
L3E228	EE09	L3E229	EE0D	L3E230	EE22	L3E231	EE23
L3E232	EE37	L3E233	EE39	L3E234	EE44	L3E235	EE5B
L3E326	EE6A	L3E327	EE87	L3E328	EE8D	L3E335	EEC9
L3E336	EECF	L3E337	EED5	L3E338	EED8	L3E339	EEDE
L3E340	EEE1	L3E341	EEE4	L3E343	EF30	L3E344	EF35
L3E345	EF3E	L3E346	EF44	L3E347	EF48	L3E348	EF61
L3E349	EF63	L3E350	EF74	L3E351	EF83	L3E352	EF8A
L3E353	EF90	L3E354	EFB8	L3E355	EFA7	L3E356	EF53
L3E357	EFBB	L3E358	EFC0	L3E359	EFD2	L3E360	EFE7
L3E361	EFF4	L3E405	EE9C	L3E406	EEA8	RHEXK	EF0F

```
002                     *
003                     *
004                     *
005                     *   ============================
006                     *** UPDATES in BASIC version V1.1 ***
007                     *   ============================
008                     *
009                     * The differences in BASIC V1.1 from V1.0 are given.
010                     *
011                     *   ]      : Indicates a total change of instruction
012                     *   !      : Indicates only a corrected jumpaddress
013                     *            due to a move of the original routine.
014                     *   No mark: Instructions only moved to other
015                     *            memory addresses (Mostly to obtain
016                     *            space for new routines).
017                     *
018                     *
019                     ********************
020                     * INTEGER COMPARE *
021                     ********************
022                     *
023                     * The temporary storage register E (sign byte) is
024                     * cleared to avoid problems when using the combined
025                     * FPT/INT exit LC27. This is done to remove a bug in
026                     * comparing 2 relational numeric operations. The
027                     * unitary operator '-' is now recognised correctly.
028                     *
029                             ORG   :C0B4
030 C0B4 C3C8D1              JMP   :D1C8     ] 
031                     *
032                             ORG   :D1C8
033 D1C8 7B                  MOV   A,E       ] Get signbyte in A
034 D1C9 1E00                MVI   E,:00     ] Clear E
035 D1CB F28CC0              JP    :C08C     ] Compare if both nrs same
036                                           sign
037 D1CE C3A4C0              JMP   :C0A4     ] Else: abort
038                     *
039                     *********
040                     * RESET *
041                     *********
042                     *
043                     * - Now switches off the volume of sound channel 2
044                     *   during initialisation. This seems useless,
045                     *   because in C76C all sound is already switched
046                     *   off.
047                     * - The header is changed to 'BASIC V1.1'
048                     *
049                             ORG   :C789
050 C789 32E401              STA   :01E4     ]  Volume SCB2 = 0
051 C78C 328F02              STA   :028F
052 C78F 117502              LXI   D,:0275
053 C792 218F02              LXI   H,:028F
054 C795 CD7CDE              CALL  :DE7C
055 C798 FB                  EI
056 C799 CF                  RST   1
057 C79A 15                  DATA  :15
058                     *
059                             ORG   :C7DD
060 C7DD 31                  DATA  :31       ] Header changed to V1.1
061                     *
062                     *
063                     *
```

```
                        **********************
                        * START FROM SCRATCH *
                        **********************
                        *
                        * Added is the reset of all keyboard pointers at
                        * various moments:
                        * - When restarting the Basic interpreter.
                        * - After command execution. Then also the output
                        *   direction is reset to screen (to avoid a not
                        *   useable keyboard).
                        * - Idem before executing 'break'.
                        * Resetting the keyboard pointers clears the key
                        * input buffer in order to avoid keybounce.
                        *
                                ORG   :C827
C827 CD63D5                     CALL  :D563     ] Keyb. pntrs to default
C82A 210000                     LXI   H,:0000
C82D 220001                     SHLD  :0100
C830 220401                     SHLD  :0104
C833 221301                     SHLD  :0113
C836 7C                         MOV   A,H
C837 322201                     STA   :0122
                        *
                                ORG   :C897
C897 21C402                     LXI   H,:02C4   ]
C89A 7E                         MOV   A,M       ] Get break flag
C89B B7                         ORA   A
C89C CA7FC8                     JZ    :C87F
C89F CD63D5                     CALL  :D563     ] Keyb.pntrs to default
C8A2 AF                         XRA   A         ]
C8A3 323101                     STA   :0131     ] Output to screen
                        *
                                ORG   :C8AC
C8AC CA9FC8                     JZ    :C89F     ] Keyb.pntrs to default,
                                                output to screen
                        *
                        ***************************
                        * EMERGENCY STOP ROUTINE *
                        ***************************
                        *
                        * No change in execution, only using a new entry on
                        * 'Run NEW'.
                        *
                                ORG   :CA25
CA25 CDB0DE                     CALL  :DEB0     Run NEW with default heap
                        *
                        ***********
                        * RUN NEW *
                        ***********
                        *
                        * - An additional entry is made for run NEW with
                        *   returning the heapsize to its default value.
                        *   In BASIC V1.0, this entry was available on
                        *   #CEC6.
                        * - All jumps/calls to RNEW with other heapsizes
                        *   are updated.
                        *
                                ORG   :DEB0
DEB0 210001                     LXI   H,:0100   ]
DEB3 229D02                     SHLD  :029D     ] Set heap to default size
DEB6 00                         NOP
DEB7 00                         NOP             Into RNEW
```

```
126                     *
127                             ORG   :C778
128 C778 CDB8DE                 CALL  :DEB8     !
129                     *
130                             ORG   :CF02
131 CF02 B8DE                   DBL   :DEB8     !
132                     *
133                             ORG   :D2A9
134 D2A9 CDB8DE                CALL  :DEB8     !
135                     *
136                     ************
137                     * RUN SAVE *
138                     ************
139                     *
140                     * Instructions only moved.
141                     *
142                             ORG   :D257
143 D257 3E30                   MVI   A,:30
144 D259 CDC502                 CALL  :02C5
145 D25C E1                     POP   H
146 D25D D1                     POP   D
147 D25E CDC802                 CALL  :02C8
148 D261 D1                     POP   D
149 D262 C3D8D7                 JMP   :D7D8
150                     *
151                     ************
152                     * RUN LOAD *
153                     ************
154                     *
155                     * Instructions only moved.
156                     *
157                             ORG   :D274
158 D274 CD23CB                 CALL  :CB23
159 D277 CD91E7                 CALL  :E791
160 D27A C5                     PUSH  B
161                     *
162                             ORG   :CF16
163 CF16 74D2                   DBL   :D274     !
164                     *
165                     ************
166                     * RUN CHECK *
167                     ************
168                     *
169                     * CALL :D7E6 is replaced by the contents of D7E6.
170                     * No change in execution.
171                     *
172                             ORG   :D2C9
173 D2C9 CDCE02                 CALL  :02CE
174 D2CC FE33                   CPI   :33
175 D2CE D2EBD2                 JNC   :D2EB
176 D2D1 0C                     INR   C
177 D2D2 3E00                   MVI   A,:00     ] Old contents
178 D2D4 CDD702                 CALL  :02D7     ] D7E6
179 D2D7 00                     NOP
180                     *
181                     ******************************
182                     * PRINT MESSAGE ON NEW LINE *
183                     ******************************
184                     *
185                     * New routine. It moves the cursor to a new line
186                     * before a message is printed. Used for printing
                        * 'STOPPED IN LINE ...' and 'END PROGRAM'.
```

```
188                     *
189                             ORG     :D7E6
190 D7E6 CD55DD                 CALL    :DD55       ] Cursor to column 0
191 D7E9 C3FFDA                 JMP     :DAFF       ] Print message
192
193                     * RSTOP:
194
195                             ORG     :DF03
196 DF03 CDE6D7                 CALL    :D7E6       ]
197
198                     * REND:
199
200                             ORG     :DF0C
201 DF0C CDE6D7                 CALL    :D7E6       ]
202                     *
203                     ******************
204                     * FAST PRINTING *
205                     ******************
206                     *
207                     * Instruction only moved.
208                     *
209                             ORG     :D7EC
210 D7EC E5                     PUSH    H
211                     *
212                             ORG     :D3CC
213 D3CC C4ECD7                 CNZ     :D7EC       !
214                     *
215                             ORG     :D78D
216 D78D C3ECD7                 JMP     :D7EC       !
217                     *
218                     *************************
219                     * READ BLOCK FROM TAPE *
220                     *************************
221                     *
222                     * Now cassette motors are switched off directly
223                     * after reading is finished.
224                     * Especially in LOADA, switching off the cassette
225                     * motors after re-arranging all data was very
226                     * annoying.
227                     *
228                             ORG     :D89A
229 D89A C365D2                 JMP     :D265       ]
230                     *
231                             ORG     :D265
232 D265 D2B3D2                 JNC     :D2B3       ] Evt run loading error
233 D268 C3D402                 JMP     :02D4       ] Stop cassette motors
234                     *
235                     ********************
236                     * INPUT TEXT LINE *
237                     ********************
238                     *
239                     * Added: Output back to screen if break pressed.
240                     * This avoids a not-useable keyboard on return.
241                     *
242                             ORG     :DD2D
243 DD2D DA36CD                 JC      :CD36       ] Jump if break pressed
244                     *
245                             ORG     :CD36
246 CD36 AF                     XRA     A           ] A=0
247 CD37 C35AD5                 JMP     :D55A       ]
248 CD3A FF                     DATA    :FF         ]
249 CD3B FF                     DATA    :FF         ]
```

```
250                     *
251                             ORG     :D55A
252 D55A 323101                 STA     :0131       ] Output to screen
253 D55D C349DD                 JMP     :DD49       ] Ignore line if break
254                                                 pressed
255                     *
256                     ********************
257                     * MULTIPLY HL * A *
258                     ********************
259                     *
260                     * Routine only moved.
261                     *
262                             ORG     :DE9C
263 DE9C DAA9DE                 JC      :DEA9       !
264 DE9F B7                     ORA     A
265 DEA0 CAACDE                 JZ      :DEAC       !
266 DEA3 EB                     XCHG
267 DEA4 29                     DAD     H
268 DEA5 EB                     XCHG
269 DEA6 D296DE                 JNC     :DE96
270 DEA9 D1                     POP     D
271 DEAA F1                     POP     PSW
272 DEAB C9                     RET
273 DEAC D1                     POP     D
274 DEAD F1                     POP     PSW
275 DEAE 3F                     CMC
276 DEAF C9                     RET
277                     *
278                     ************
279                     * RUN RUN *
280                     ************
281                     *
282                     * The sequence of the instructions is modified.
283                     * This enables RUN <linenr> to be used without
284                     * destroying the contents of heap and symtab.
285                     *
286                             ORG     :DF9E
287 DF9E CD23CB                 CALL    :CB23       Empty heap and symtab
288 DFA1 2A9F02                 LHLD    :029F       Get start textbuf
289 DFA4 44                     MOV     B,H         ) in BC
290 DFA5 4D                     MOV     C,L         )
291 DFA6 CD01E4                 CALL    :E401       Run RESTORE
292 DFA9 210000                 LXI     H,:0000
293 DFAC 221501                 SHLD    :0115       Reset step/trace flag
294 DFAF AF                     XRA     A
295 DFB0 322601                 STA     :0126       No suspended program
296 DFB3 3100F9                 LXI     SP,:F900    Reset SP
297
298                     * RUN <linenr> :
299
300                             ORG     :DFBD
301 DFBD C3A4DF                 JMP     :DFA4       ! Process RUN
302                     *
303                     ******************************
304                     * STORE DATA (READ, INPUT) *
305                     ******************************
306                     *
307                     * Handle string inputs (no change):
308                     *
309                             ORG     :E385
310 E385 CCB8E4                 CZ      :E4B8       ] Only JZ :E3A2 replaced
311 E388 C4B4E4                 CNZ     :E4B4       ]       by its contents
```

```
312
313                 * Error exit changed: Now checks if end of input
314                 * is reached. EFEPT is only updated if not running
315                 * inputs (anymore).
316
317                         ORG   :E39C
318 E39C CA67E3            JZ    :E367      ] Read next data if no error
319 E39F 3A1701            LDA   :0117      If error: Get 'run-input'
320                                          flag
321 E3A2 CDC3E3            CALL  :E3C3      ]
322 E3A5 C30BDA            JMP   :DA0B      ] Run 'SYNTAX ERROR'
323                 *
324                         ORG   :E3C3
325 E3C3 A7                ANA   A          ] Set flags
326 E3C4 C0                RNZ              ] Abort if running inputs
327 E3C5 2A3201            LHLD  :0132      Get EFEPT
328 E3C8 11FCFF            LXI   D,:FFFC
329 E3CB 19                DAD   D          -4
330 E3CC 220001            SHLD  :0100      Set start current line
331 E3CF C9                RET
332                 *
333                 ****************************
334                 * PREPARE GETTING INPUTS *
335                 ****************************
336                 *
337                 * The keyboard pointers are set to their default
338                 * values before inputs are asked. This avoids
339                 * keybounce.
340                 *
341                         ORG   :E447
342 E447 221D01            SHLD  :011D
343 E44A CD63D5            CALL  :D563      ] Init keyb pntrs
344 E44D 3EFF              MVI   A,:FF
345 E44F 321701            STA   :0117
346 E452 C3D0E3            JMP   :E3D0
347                 *
348                 ******************
349                 * cont. of 0E401 *
350                 ******************
351                 *
352                 * Instruction only moved.
353                 *
354 E455 322301            STA   :0123
355 E458 C9                RET
356 E459 FF                DATA  :FF
357                 *
358                         ORG   :E409
359 E409 C355E4            JMP   :E455      !
360                 *
361                 *************
362                 * RUN CLEAR *
363                 *************
364                 *
365                 * Routine is completely modified.
366                 * Max. useable heap space in V1.0 was #7FFF-4.
367                 * Now it is #7FFF.
368                 * Doesnot set up anymore a complete new heap, but
369                 * just empties heap and symboltable entries and
370                 * shifts the program to after the new heap.
371                 *
372                         ORG   :E6B5
373 E6B5 CDF8E6            CALL  :E6F8      ] Get reqd space in HL
```

```
374 E6B8 E5                 PUSH  H        ] Preserve it
375 E6B9 7C                 MOV   A,H      ] Get hibyte in A
376 E6BA 2B                 DCX   H        ]
377 E6BB 2B                 DCX   H        ]
378 E6BC 2B                 DCX   H        ]
379 E6BD 2B                 DCX   H        ] Reqd space -4
380 E6BE B4                 ORA   H        ]
381 E6BF FA15DA             JM    :DA15    ] Run 'NUMBER OUT OF RANGE'
382                                        error if < 4 or > 32K.
383 E6C2 D1                 POP   D        ] Get reqd space in DE
384 E6C3 2A9D02             LHLD  :029D    ] Get old heapsize
385 E6C6 EB                 XCHG           ]
386 E6C7 229D02             SHLD  :029D    ] Store new heapsize
387 E6CA C314D2             JMP   :D214    ]
388 E6CD FF                 DATA  :FF      ]
389                 *
390                         ORG   :D214
391 D214 CD1ADE             CALL  :DE1A    ] Calc difference old/new
392 D217 E5                 PUSH  H        ] Preserve result
393 D218 CD23CB             CALL  :CB23    ] Empty heap and symtab,
394                                        move program to after heap
395 D21B 2A0001             LHLD  :0100    Get pntr to current line
396 D21E 7C                 MOV   A,H
397 D21F B5                 ORA   L
398 D220 D1                 POP   D        Get difference
399 D221 C8                 RZ             Abort if direct cmd
400 D222 19                 DAD   D        ] Add difference to pntr
401                                        current line
402 D223 C3BBCE             JMP   :CEBB    ]
403 D226 FF                 DATA  :FF      ]
404                 *
405                         ORG   :CEBB
406 CEBB 220001             SHLD  :0100    ] Update pntr
407 CEBE CD01E4             CALL  :E401    ] Run RESTORE
408 CEC1 D5                 PUSH  D        ] Preserve difference
409 CEC2 C37FD8             JMP   :D87F    ]
410                 *
411                         ORG   :D87F
412 D87F E1                 POP   H        ] Get difference
413 D880 09                 DAD   B        ] Add textpntr
414 D881 44                 MOV   B,H      ] New textpntr in BC
415 D882 4D                 MOV   C,L      ]
416 D883 B7                 ORA   A        ] No special action
417 D884 C9                 RET            ]
418 D885 FF                 DATA  :FF      ]
419                 *
420                 ****************************
421                 * RUN A VARIABLE POINTER *
422                 ****************************
423                 *
424                 * This enables the use of DIM (255,255). In V1.0,
425                 * the max. dimension was 254.
426                 *
427                         ORG   :E9BE
428 E9BE CDF0ED             CALL  :EDF0    ] Another calculation
429                                        routine is used
430                 *
431                 ************
432                 * RUN GETC *
433                 ************
434                 *
435                 * Better entry to keyboard scan routine to
```

```
436                     * avoid keybounce.
437                     *
438                             ORG   :EB75
439 EB75 AF                     XRA   A          ]
440 EB76 32B902                 STA   :02B9      ] Clear breakflag
441 EB79 CDBED6                 CALL  :D6BE      ] Run GETC
442                     *
443                             ORG   :EA04
444 EA04 756B                   DBL   :6B75      !
445                     *
446                     ***********
447                     * RUN TAB *
448                     ***********
449                     *
450                     * The old routine worked only for DOUTC=0. Now
451                     * DOUTC is not checked anymore, but the Z-flag
452                     * set by CE60 is evaluated. If Z=1 (all O.K.),
453                     * the TAB-instruction is executed. If Z=0, then
454                     * the number of tab's is not correct. Then only
455                     * one space is printed.
456                     *
457                             ORG   :EAA5
458 EAA5 1F                     RAR              ] Dummy to preserve Z-flag
459                     *
460                     ***********
461                     * RUN INT *
462                     ***********
463                     *
464                     * Running INT on a number with value 0 gave as
465                     * result: -1. Now this failure is corrected.
466                     *
467                             ORG   :EB90
468 EB90 212901                 LXI   H,:0129    ]
469 EB93 E7                     RST   4          ] Copy MACC to WORKE
470 EB94 0F                     DATA  :0F        ]
471 EB95 C1                     POP   B          ]
472 EB96 E7                     RST   4          ] MACC = INT (MACC)
473 EB97 1E                     DATA  :1E        ] for FPT number
474 EB98 B7                     ORA   A          ] Set flags on exp byte
475 EB99 F0                     RP               ] Ready if nr positive
476 EB9A C3C5CE                 JMP   :CEC5      ]
477                     *
478                             ORG   :CEC5
479 CEC5 CD0CC0                 CALL  :C00C      ] FPT compare nrs in MACC
480                                              and in WORKE
481 CEC8 21F1D0                 LXI   H,:D0F1    ] Addr FPT (-1)
482 CECB C8                     RZ               ] Ready if nr = 0
483 CECC E7                     RST   4          ] Add (-1) to MACC if nr <0
484 CECD 00                     DATA  :00        ]
485 CECE C9                     RET              ]
486                     *
487                     ********************
488                     * part of RUN TALK *
489                     ********************
490                     *
491                     * A bug is removed, caused by XCHG in #EC71. The
492                     * return from #EC78 (JMP CD67) could not be executed
493                     * correctly, because of a wrong address stored in
494                     * the HL-registers. This problem existed only for
495                     * the 'TALK'-codes #0C (wait) and #0D (ML call).
496                     *
497                     * Entry: DE: Wait time or address ML-routine.
```

```
498                     *
499                             ORG   :EC71
500 EC71 CCCCDA                 CZ    :DACC     ] If to be waited
501 EC74 C462CD                 CNZ   :CD62     ] Goto MLP address
502 EC77 C367CD                 JMP   :CD67     Handle next code
503 EC7A FF                     DATA  :FF
504
505                     * If wait:
506                     * As old routine, but HL replaced by DE.
507
508                             ORG   :DACC
509 DACC 1B                     DCX   D         ]
510 DACD 7A                     MOV   A,D       ]
511 DACE B3                     ORA   E         ]
512 DACF C2CCDA                 JNZ   :DACC     ] Wait for DE=0
513 DAD2 C9                     RET             ]
514 DAD3 FF                     DATA  :FF       ]
515
516                     * To enable ML call:
517
518                             ORG   :CD62
519 CD62 D5                     PUSH  D         ] Address MLP on stack
520 CD63 C9                     RET             ] Go to it
521                     *
522                     ************************
523                     * TEST A FPT VARIABLE *
524                     ************************
525                     *
526                     * The old routine tested all bytes of the FPT nr.
527                     * Now only the exponent byte and the hibyte of the
528                     * mantissa is tested. In this way, very small FPT
529                     * numbers are considered to be zero.
530                     *
531                             ORG   :EC91
532 EC91 CA98EC                 JZ    :EC98     ] Abort if A and B are 0
533 EC94 7B                     MOV   A,E       ] Exp byte in A
534 EC95 F601                   ORI   :01       ] Clear CY-flag
535 EC97 00                     NOP             ]
536                     *
537                     *
538                     *    ============
539                     ***** ROM BANK 1 *****
540                     *    ============
541                     *
542                     *
543                     ************
544                     * FPT EXP *
545                     ************
546                     *
547                     * Test for overflow is modified. Overflow occurs
548                     * for e^X when -45 < X < 43.6.
549                     * This is checked now before exponent routine is
550                     * entered.
551                     *
552                             ORG   :E688
553 E688 CAC9EF                 JZ    :EFC9     ]
554                     *
555                             ORG   :EFC9
556 EFC9 7A                     MOV   A,D       ] Get lobyte MACC
557 EFCA E6C0                   ANI   :C0       ] Check for max. value
558 EFCC C28BE6                 JNZ   :E68B     ] Jump if nr too big
559 EFCF C394E6                 JMP   :E694     ] If O.K.
```

```
560                     *
561                     *********
562                     * LOADA *
563                     *********
564                     *
565                     * Switching off the cassette motors is now part of
566                     * D897. So the cassette motors are switched off
567                     * directy after the reading from tape is done.
568                     *
569                             ORG   :EE6B
570 EE6B C332EE                 JMP   :EE32     ] Adapted to new situation
571                     *
572                     *
573                     *    ============
574                     ***** ROM BANK 2 *****
575                     *    ============
576                     *
577                     *
578                     ***************
579                     * WINDOW DOWN *
580                     ***************
581                     *
582                     * A bug which let the cursor disappear sometimes is
583                     * removed.
584                     *
585                             ORG   :ECB6
586 ECB6 2AA900                 LHLD  :00A9     ] Get offset top of window
587                                             from start buffer
588 ECB9 EB                     XCHG            ] in DE
589 ECBA CD6BD2                 CALL  :D26B     ]
590 ECBD FCABED                 CM    :EDAB     Then cursor down
591                     *
592                             ORG   :D26B
593 D26B 2AAC00                 LHLD  :00AC     ] Get Y-offset cursor in
594                                             document
595 D26E CDF2E6                 CALL  :E6F2     ] HL=HL-DE
596 D271 2D                     DCR   L         ] -1
597 D272 37                     STC             ] Set 'window changed' flag
598 D273 C9                     RET             ]
599                     *
600                     *
601                     *    ============
602                     ***** ROM BANK 3 *****
603                     *    ============
604                     *
605                     *
606                     ****************************************
607                     * ERROR EXIT 'ENCODE INT NR INTO EBUF *
608                     ****************************************
609                     *
610                             ORG   :E8B7
611 E8B7 219FE3                 LXI   H,:E39F   ] (0) Addr routine 'STORE
612                                             DATA'
613 E8BA E5                     PUSH  H         ] Preserve it as returnaddr
614 E8BB 3A4000                 LDA   :0040     ] Get POROM
615 E8BE E63F                   ANI   :3F       ] Select bank 0
616 E8C0 F5                     PUSH  PSW       ]
617 E8C1 C3E6C6                 JMP   :C6E6     ] Bank return
618                     *
619                     *
620                     *
621 E8C4                        END
```