```
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
```