```
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
```

```
250 E325 C34DC1             JMP   :C14D     Popall, ret
251
252                 * If O.K.:
253
254 E328 CDCFE3     L1E37   CALL  :E3CF     Copy reg BCDE into MACC
255 E32B C34DC1             JMP   :C14D     Popall, ret
256                 *
257                 ********
258                 * IAND *
259                 ********
260                 *
261                 * Logical 'AND': MACC = MACC IAND MEM.
262                 *
263                 * Entry: HL points to operand in memory.
264                 * Exit:  All registers preserved.
265                 *
266 E32E F5          XIAND   PUSH  PSW
267 E32F C5                  PUSH  B
268 E330 D5                  PUSH  D
269 E331 E5                  PUSH  H
270 E332 C3E1EC            JMP   :ECE1     Prepare IAND and perform
271                 *
272                 ****************
273                 * PERFORM IAND *
274                 ****************
275                 *
276                 * Performs: MEM IAND EBCA, result in ABCD.
277                 *
278                 * Entry: HL points to last byte of MEM.
279                 *         Fixed point number in EBCA.
280                 * Exit:  HL points to 1st byte of MEM.
281                 *         E preserved.
282                 *
283 E335 A6          SIAND   ANA   M
284 E336 57                  MOV   D,A
285 E337 2B                  DCX   H
286 E338 79                  MOV   A,C
287 E339 A6                  ANA   M
288 E33A 4F                  MOV   C,A
289 E33B 2B                  DCX   H
290 E33C 78                  MOV   A,B
291 E33D A6                  ANA   M
292 E33E 47                  MOV   B,A
293 E33F 2B                  DCX   H
294 E340 7B                  MOV   A,E
295 E341 A6                  ANA   M
296 E342 C9                  RET
297                 *
298 E343 85E3       L1E40   DBL   :E385     (not used)
299                 *
300                 *******
301                 * IOR *
302                 *******
303                 *
304                 * Logical 'OR': MACC = MACC IOR MEM.
305                 *
306                 * Entry: HL points to operand in memory.
307                 * Exit:  All registers preserved.
308                 *
309 E345 F5          XIOR    PUSH  PSW
310 E346 C5                  PUSH  B
311 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		