```
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.
187                *
```



```
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         )
```



```
E17C 78                 MOV   A,B       )
E17D 8E                 ADC   M         )
E17E 47                 MOV   B,A       )
E17F 2B                 DCX   H         )
E180 7B                 MOV   A,E       )
E181 8E                 ADC   M         )
E182 5F                 MOV   E,A       )
E183 1F                 RAR             Evt carry in msb
E184 AB                 XRA   E         Msb=1 if overflow
E185 E1                 POP   H         Get compl 00D5 EXOR M
                                        (0 if different signbits)
E186 A4                 ANA   H         Overflow only if different
                                        signbits
E187 FC4BC0     L1E11   CM    :C04B     Then run overflow error
E18A C384E3             JMP   :E384     Copy E into A; Then reg
                                        ABCD into MACC
                        *
                        ************************
                        * INTEGER SUBTRACTION *
                        ************************
                        *
                        * Signed 32-bit subtraction: MACC = MACC - MEM.
                        * Evt. overflow handling via C04B.
                        *
                        * Entry: HL: Points to 1st byte of operand.
                        * Exit:  All registers preserved.
                        *
E18D F5         XISUB   PUSH  PSW
E18E C5                 PUSH  B
E18F D5                 PUSH  D
E190 E5                 PUSH  H
E191 CD8CE3             CALL  :E38C     Copy MACC into reg EBCA
                                        D = compl 00D5 EXOR M
E194 D5                 PUSH  D
E195 96                 SUB   M         )
E196 57                 MOV   D,A       )
E197 2B                 DCX   H         )
E198 79                 MOV   A,C       )
E199 9E                 SBB   M         )
E19A 4F                 MOV   C,A       ) Subtract EBCA - MEM
E19B 2B                 DCX   H         )  Result in EBCD
E19C 78                 MOV   A,B       )
E19D 9E                 SBB   M         )
E19E 47                 MOV   B,A       )
E19F 2B                 DCX   H         )
E1A0 7B                 MOV   A,E       )
E1A1 9E                 SBB   M         )
E1A2 5F                 MOV   E,A       )
E1A3 1F                 RAR             Evt carry in msb
E1A4 AB                 XRA   E         Msb=1 if overflow
E1A5 E1                 POP   H         Get compl 00D5 EXOR M
E1A6 B4                 ORA   H
E1A7 2F                 CMA
E1A8 B7                 ORA   A         Msb=1 ?
E1A9 C387E1             JMP   :E187     Evt run overflow error;
                                        Copy result into MACC
                        *
                        ************************
                        * INTEGER MULTIPLICATION *
                        ************************
                        *
                        * Signed 32-bit multiplication: MACC = MACC * MEM.
```

```
                        * The overflow exit is taken if both factors are
                        * more than 2 bytes or the product is longer than
                        * the signbit of the 4th byte.
                        * If MEM > 2 bytes, than MACC into DE and MEM into
                        * MACC, assuming that MACC is max. 2 bytes.
                        *
                        * Entry: HL: Points to multiplier.
                        * Exit:  All registers preserved.
                        *
E1AC F5         XIMUL   PUSH  PSW
E1AD C5                 PUSH  B
E1AE D5                 PUSH  D
E1AF E5                 PUSH  H
E1B0 3AD500             LDA   :00D5     Get sign byte
E1B3 B7                 ORA   A
E1B4 FC15E3             CM    :E315     If nr <0: change sign
E1B7 DA25E2             JC    :E225     (Don't work: E315 preserves
                                        flags; ORA A clears CY)
E1BA AE                 XRA   M         Final sign bit in S-flag
E1BB F5                 PUSH  PSW       Save it
E1BC 7E                 MOV   A,M       )
E1BD 23                 INX   H         )
E1BE B7                 ORA   A         ) Get MEM into BCDE
E1BF 47                 MOV   B,A       ) Set flags on sign byte
E1C0 4E                 MOV   C,M       )
E1C1 23                 INX   H         )
E1C2 56                 MOV   D,M       )
E1C3 23                 INX   H         )
E1C4 5E                 MOV   E,M       )
E1C5 FCC9E3             CM    :E3C9     If MEM <0: negate BCDE
E1C8 DA25E2             JC    :E225     Error exit if overflow
E1CB B1                 ORA   C
E1CC CADFE1             JZ    :E1DF     Jump if MEM <= 2 bytes

                        * MEM > 2 bytes: exchange MACC with BCDE:

E1CF 2AD500             LHLD  :00D5     Get hibytes MACC
E1D2 7C                 MOV   A,H
E1D3 B5                 ORA   L
E1D4 C225E2             JNZ   :E225     Overflow exit if MACC >
                                        2 bytes
E1D7 2AD700             LHLD  :00D7     Get lobytes MACC in HL
E1DA CDCFE3             CALL  :E3CF     Copy reg BCDE (=MEM) into
                                        MACC
E1DD 55                 MOV   D,L       )
E1DE 5C                 MOV   E,H       ) Orig. MACC into DE

                        * Now 4-byte nr in MACC and 2 byte nr in DE:

E1DF 210000     L1E14   LXI   H,:0000
E1E2 E5                 PUSH  H         0000 on stack
E1E3 21D800             LXI   H,:00D8   Addr lobyte MACC
E1E6 4E                 MOV   C,M       lobyte in C
E1E7 E3         L1E15   XTHL            On stack: addr current MACC
                                        byte
E1E8 79                 MOV   A,C       Current byte in A
E1E9 B7                 ORA   A
E1EA CA1EE2             JZ    :E21E     Jump if byte=0
E1ED 0680               MVI   B,:80
E1EF 79         L1E16   MOV   A,C       ) Current MACC byte in A
E1F0 1F                 RAR             )
E1F1 4F                 MOV   C,A       )
```



```
E1F2 D2F6E1             JNC   :E1F6     ) SHR reg H,L and B as long
                                        ) no carry from C
E1F5 19                 DAD   D         ) Else: Add other nr to HL
E1F6 7C         L1E17   MOV   A,H       )
E1F7 1F                 RAR             )
E1F8 67                 MOV   H,A       )
E1F9 7D                 MOV   A,L       )-- Effect: Multiply MACC by
E1FA 1F                 RAR             )           DE, result in B
E1FB 6F                 MOV   L,A       )
E1FC 78                 MOV   A,B       )
E1FD 1F                 RAR             )
E1FE 47                 MOV   B,A       )
E1FF D2EFE1             JNC   :E1EF     ) Do L1E16 max 8 times
E202 E3                 XTHL            HL pnts to current MACC byte
E203 70                 MOV   M,B       Result in MACC byte
E204 2B                 DCX   H         Pnts to next lower MACC byte
E205 7D                 MOV   A,L       Get lobyte of addr
E206 FE04               CPI   :04       Ready?
E208 4E                 MOV   C,M       Get next lower byte in C
E209 C2E7E1             JNZ   :E1E7     Not ready: mult. next byte

                        * Multiplication done:

E20C E1                 POP   H
E20D 78                 MOV   A,B       Get last result
E20E B7                 ORA   A
E20F FA25E2             JM    :E225     Error exit if overflow
E212 7C                 MOV   A,H       )
E213 B5                 ORA   L         ) Check if HL<>0
E214 C225E2             JNZ   :E225     Then overflow exit
E217 F1         L1E19   POP   PSW       Get final sign in S-flag
E218 FC15E3             CM    :E315     Change sign MACC if nr must
                                        be negative
E21B C34DC1             JMP   :C14D     Popall, ret

                        * If MACC byte = 0:

E21E 45         L1E20   MOV   B,L       ) Move bytes in HLB one byte
E21F 6C                 MOV   L,H       ) down
E220 2600               MVI   H,:00     )
E222 C302E2             JMP   :E202     Go to next MACC byte

                        * If overflow error:

E225 CD4BC0     L1E21   CALL  :C04B     Run overfow error
E228 C317E2             JMP   :E217     Quit
                        *
                        *
                        *
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
125 E2AA 17                 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
```

```
E348 E5              PUSH  H
E349 C3EAEC          JMP   :ECEA     Prepare IOR and perform
                     *
                     ****************
                     * PERFORM IOR *
                     ****************
                     *
                     * Performs MEM IOR EBCA. Result in ABCD.
                     *
                     * Entry: HL points to last byte of MEM.
                     *        Fixed point nr in EBCA.
                     * Exit:  HL points to 1st byte of MEM.
                     *        E preserved.
                     *
E34C B6              SIOR  ORA   M
E34D 57                    MOV   D,A
E34E 2B                    DCX   H
E34F 79                    MOV   A,C
E350 B6                    ORA   M
E351 4F                    MOV   C,A
E352 2B                    DCX   H
E353 78                    MOV   A,B
E354 B6                    ORA   M
E355 47                    MOV   B,A
E356 2B                    DCX   H
E357 7B                    MOV   A,E
E358 B6                    ORA   M
E359 C9                    RET
                     *
E35A 85E3            L1E43 DBL   :E385     (not used)
                     *
                     ********
                     * IXOR *
                     ********
                     *
                     * Logical 'XOR': MACC = MACC IXOR MEM.
                     *
                     * Entry: HL points to operand in memory.
                     * Exit:  All registers preserved.
                     *
E35C F5              XIXOR PUSH  PSW
E35D C5                    PUSH  B
E35E D5                    PUSH  D
E35F E5                    PUSH  H
E360 C3F3EC                JMP   :ECF3     Prepare IXOR and perform
                     *
                     ****************
                     * PERFORM IXOR *
                     ****************
                     *
                     * Performs MEM IXOR EBCA, result in ABCD.
                     *
                     * Entry: HL points last byte of MEM.
                     *        Fixed point number in EBCA.
                     * Exit:  HL points to 1st byte of MEM.
                     *        E preserved.
                     *
E363 AE              SIXOR XRA   M
E364 57                    MOV   D,A
E365 2B                    DCX   H
E366 79                    MOV   A,C
E367 AE                    XRA   M
```



```
E368 4F                    MOV   C,A
E369 2B                    DCX   H
E36A 78                    MOV   A,B
E36B AE                    XRA   M
E36C 47                    MOV   B,A
E36D 2B                    DCX   H
E36E 7B                    MOV   A,E
E36F AE                    XRA   M
E370 C9                    RET
                     *
E371 85E3            L1E46 DBL   :E385     (not used)
                     *
                     ********
                     * INOT *
                     ********
                     *
                     * Logical 'INOT': MACC = INOT (MACC).
                     *
                     * Exit: All registers preserved.
                     *
E373 F5              XINOT PUSH  PSW
E374 C5                    PUSH  B
E375 D5                    PUSH  D
E376 E5                    PUSH  H
E377 CDFCEC                CALL  :ECFC     Copy MACC into ABCD; CMA
E37A 5F                    MOV   E,A       Save hibyte
E37B 7A                    MOV   A,D
E37C 2F                    CMA             INOT D
E37D 57                    MOV   D,A
E37E 79                    MOV   A,C
E37F 2F                    CMA             INOT C
E380 4F                    MOV   C,A
E381 78                    MOV   A,B
E382 2F                    CMA             INOT B
E383 47                    MOV   B,A
E384 7B              L1E48 MOV   A,E       Get back hibyte
E385 CD26E1          L1E49 CALL  :E126     Copy ABCD into MACC
E388 C34DC1                JMP   :C14D     Popall, ret
                     *
E38B C9              L1E50 RET             (not used)
                     *
                     ************************************
                     * COPY MACC INTO REG. E,B,C,A      *
                     * D = compl. OOD5 EXOR hibyte MEM *
                     ************************************
                     *
                     * Entry: HL points to 1st byte MEM.
                     * Exit:  HL points to last byte MEM.
                     *        D = complement EXOR hibytes MACC and MEM.
                     *            Msb D = 1: sign bits identical.
                     *            Msb D = 0: sign bits different.
                     *
E38C C301ED          L1E51 JMP   :ED01     Copy MACC into ABCD, A in E:
                                           jump to E38F
                     *
E38F AE              L1E52 XRA   M         EXOR sign bytes
E390 2F                    CMA             Complement result
E391 23                    INX   H
E392 23                    INX   H
E393 23                    INX   H
E394 D5                    PUSH  D
E395 57                    MOV   D,A       A = compl EXOR signbytes
```

```
436 E396 F1                 POP   PSW       Get D in A
437 E397 C9                 RET
438                 *
439                 *******
440                 * SHR *
441                 *******
442                 *
443                 * MACC = MACC SHR MEM.
444                 * Shifts contents MACC right (MEM) places.
445                 *
446                 * Entry: HL points to operand in memory.
447                 * Exit:  All registers preserved.
448                 *        Result is 0 if MEM < 0 or > 31.
449                 *
450 E398 F5          XSHR    PUSH  PSW
451 E399 C5                  PUSH  B
452 E39A D5                  PUSH  D
453 E39B E5                  PUSH  H
454 E39C CD08ED              CALL  :ED08     Check MEM. If <=31:
455                                          MACC into BCDE, shift in A
456                                          Else: clear ABCDE
457 E39F CD55EB              CALL  :EB55     Shift BCDE right A places
458 E3A2 C328E3              JMP   :E328     BCDE into MACC; quit
459                 *
460                 *******
461                 * SHL *
462                 *******
463                 *
464                 * MACC = MACC SHL MEM.
465                 * Shifts contents MACC left (MEM) places.
466                 *
467                 * Entry/exit: See XSHR.
468                 *
469 E3A5 F5          XSHL    PUSH  PSW
470 E3A6 C5                  PUSH  B
471 E3A7 D5                  PUSH  D
472 E3A8 E5                  PUSH  H
473 E3A9 CD08ED              CALL  :ED08     Check MEM. If <= 31: MACC
474                                          into BCDE, shift in A
475                                          Else: clear ABCDE
476 E3AC CD39EB              CALL  :EB39     Shift BCDE left A places
477 E3AF C328E3              JMP   :E328     BCDE into MACC; quit
478                 *
479                 ********************************
480                 * TEST VALUE OF AN INT NUMBER *
481                 ********************************
482                 *
483                 * Tests if an INT has a value between 0 and 31.
484                 * If true: Number into A, else: Clear ABCDE.
485                 *
486                 * Entry: HL points to a 4-byte number.
487                 * Exit:  Nr <= #1F: Number in A.
488                 *        Nr > #1F: ABCDE cleared.
489                 *        HL points to 4th byte in memory.
490                 *
491 E3B2 7E          XSTST   MOV   A,M       Get 1st byte
492 E3B3 23                  INX   H
493 E3B4 B6                  ORA   M         OR with 2nd
494 E3B5 23                  INX   H
495 E3B6 B6                  ORA   M         OR with 3rd
496 E3B7 23                  INX   H
497 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                *************
249                *
```



```
250 E4D1 CD35E5    ZSIN  CALL  :E535     Wait, operate immediate
251 E4D4 02              DATA  :02       MTOS = SIN (MTOS)
252 E4D5 C9              RET
253                *
254                *************
255                * AMD: COS *
256                *************
257                *
258 E4D6 CD35E5    ZCOS  CALL  :E535     Wait, operate immediate
259 E4D9 03              DATA  :03       MTOS = COS (MTOS)
260 E4DA C9              RET
261                *
262                *************
263                * AMD: TAN *
264                *************
265                *
266 E4DB CD35E5    ZTAN  CALL  :E535     Wait, operate immediate
267 E4DE 04              DATA  :04       MTOS = TAN (MTOS)
268 E4DF C9              RET
269                *
270                **************
271                * AMD: ASIN *
272                **************
273                *
274 E4E0 CD35E5    ZASIN CALL  :E535     Wait, operate immediate
275 E4E3 05              DATA  :05       MTOS = ASIN (MTOS)
276 E4E4 C9              RET
277                *
278                **************
279                * AMD: ACOS *
280                **************
281                *
282 E4E5 CD35E5    ZACOS CALL  :E535     Wait, operate immediate
283 E4E8 06              DATA  :06       MTOS = ACOS (MTOS)
284 E4E9 C9              RET
285                *
286                *************
287                * AMD: ATN *
288                *************
289                *
290 E4EA CD35E5    ZATAN CALL  :E535     Wait, operate immediate
291 E4ED 07              DATA  :07       MTOS = ATAN (MTOS)
292 E4EE C9              RET
293                *
294                ****************************************
295                * AMD: CHANGE CONTENTS MTOS TO INTEGER *
296                ****************************************
297                *
298 E4EF CD35E5    ZFIX  CALL  :E535     Wait, operate immediate
299 E4F2 1E              DATA  :1E       MTOS = INT(MTOS)
300 E4F3 C9              RET
301                *
302                **********************
303                * AMD: INT ADDITION *
304                **********************
305                *
306                * MTOS = MTOS + MEM.
307                *
308                * Entry: HL points to number in memory.
309                * Exit:  All registers reserved.
310                *
311 E4F4 CD27E5    ZIADD CALL  :E527     Wait, load, operate imm.
```

```
E4F7 2C                 DATA  :2C       INT addition
E4F8 C9                 RET
                *
                ************************
                * AMD: INT SUBTRACTION *
                ************************
                *
                * MTOS = MTOS - MEM.
                *
                * Entry: HL points to number in memory.
                * Exit:  All registers preserved.
                *
E4F9 CD27E5     ZISUB   CALL  :E527     Wait, load, operate imm.
E4FC 2D                 DATA  :2D       INT subtraction
E4FD C9                 RET
                *
                ****************************
                * AMD: INT MULTIPLICATION *
                ****************************
                *
                * MTOS = MTOS * MEM.
                *
                * Entry: HL points to number in memory.
                * Exit:  All registers preserved.
                *
E4FE CD27E5     ZIMUL   CALL  :E527     Wait, load, operate imm.
E501 2E                 DATA  :2E       INT multiplication
E502 C9                 RET
                *
                **********************
                * AMD: INT DIVISION *
                **********************
                *
                * MTOS = MTOS / MEM.
                *
                * Entry: HL points to number in memory.
                * Exit:  All registers preserved.
                *
E503 CD27E5     ZIDIV   CALL  :E527     Wait, load, operate imm.
E506 2F                 DATA  :2F       INT division
E507 C9                 RET
                *
                ****************************
                * AMD: INT DIVIDE REMAINDER *
                ****************************
                *
                * MTOS = INT remainder of MTOS / MEM.
                *
                * Entry: HL points to number in memory.
                * Exit:  All registers preserved.
                *
E508 CD35E5     ZIREM   CALL  :E535     Wait, operate immediate
E50B 37                 DATA  :37       Push MTOS
E50C CD03E5             CALL  :E503     INT divide
E50F CDFEE4             CALL  :E4FE     INT multiply back
E512 CD35E5             CALL  :E535     Wait, operate immediate
E515 2D                 DATA  :2D       Subtract: difference =
                                        remainder
E516 C9                 RET
                *
                *
                *
```



```
                ****************
                * 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
497 E58B E5                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.
125                 * 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,
```

```
                                          sign in CY
E6DD D1                 POP   D           Get addr FPT(n/8)
E6DE F5                 PUSH  PSW
E6DF 211000             LXI   H,:0010     Init offset for table L1E283
E6E2 D2E6E6             JNC   :E6E6       Jump if X was positive
E6E5 29                 DAD   H           Offset is #0020 for neg.nr.
E6E6 19         L1E124  DAD   D           Calc addr in L1E283
E6E7 CD59EA             CALL  :EA59       Calc 2^z * 2^d
E6EA F1                 POP   PSW         Get CY on sign of X
E6EB E1                 POP   H           Get n in H
E6EC 7C                 MOV   A,H
E6ED D2F2E6             JNC   :E6F2       Jump if X was positive
E6F0 2F                 CMA               ) Else: complement n
E6F1 3C                 INR   A           )
E6F2 CDB7C1     L1E125  CALL  :C1B7       Add exponents (n+d+z)
E6F5 DC4BEA     L1E126  CC    :EA4B       Evt error handling
E6F8 C34DC1     L1E127  JMP   :C14D       Popall, ret

                * CONSTANTS FOR 'XEXP':

E6FB 7E         L1E279  DATA  :7E         FPT(1/8)
E6FC 80                 DATA  :80
E6FD 00                 DATA  :00
E6FE 00                 DATA  :00
                *
E6FF 7F         L1E280  DATA  :7F         FPT(3/8)
E700 C0                 DATA  :C0
E701 00                 DATA  :00
E702 00                 DATA  :00
                *
E703 00         L1E281  DATA  :00         FPT(5/8)
E704 A0                 DATA  :A0
E705 00                 DATA  :00
E706 00                 DATA  :00
                *
E707 00         L1E282  DATA  :00         FPT(7/8)
E708 E0                 DATA  :E0
E709 00                 DATA  :00
E70A 00                 DATA  :00
                *
                *
E70B 01         L1E283  DATA  :01         2^(1/8)
E70C 8B                 DATA  :8B
E70D 95                 DATA  :95
E70E C2                 DATA  :C2
                *
E70F 01                 DATA  :01         2^(3/8)
E710 A5                 DATA  :A5
E711 FE                 DATA  :FE
E712 D7                 DATA  :D7
                *
E713 01                 DATA  :01         2^(5/8)
E714 C5                 DATA  :C5
E715 67                 DATA  :67
E716 2A                 DATA  :2A
                *
E717 01                 DATA  :01         2^(7/8)
E718 EA                 DATA  :EA
E719 C0                 DATA  :C0
E71A C7                 DATA  :C7
                *
E71B 00         L1E287  DATA  :00         2^(-1/8)
```



```
E71C EA                 DATA  :EA
E71D C0                 DATA  :C0
E71E C7                 DATA  :C7
                *
E71F 00                 DATA  :00         2^(-3/8)
E720 C5                 DATA  :C5
E721 67                 DATA  :67
E722 2A                 DATA  :2A
                *
E723 00                 DATA  :00         2^(-5/8)
E724 A5                 DATA  :A5
E725 FE                 DATA  :FE
E726 D7                 DATA  :D7
                *
E727 00                 DATA  :00         2^(-7/8)
E728 8B                 DATA  :8B
E729 95                 DATA  :95
E72A C2                 DATA  :C2
                *
                *
E72B 01         L1E291  DATA  :01         1/LN2
E72C B8                 DATA  :B8
E72D AA                 DATA  :AA
E72E 3B                 DATA  :3B
                *
                *
E72F 00         L1E292  DATA  :00         a1: LN2
E730 B1                 DATA  :B1         0.69314718057
E731 72                 DATA  :72
E732 18                 DATA  :18
                *
E733 7E                 DATA  :7E         a2: ((LN2)^2)/2!
E734 F5                 DATA  :F5         0.24022648580
E735 FD                 DATA  :FD
E736 EF                 DATA  :EF
                *
E737 7C                 DATA  :7C         a3: ((LN2)^3)/3!
E738 E3                 DATA  :E3         0.055504105406
E739 58                 DATA  :58
E73A 46                 DATA  :46
                *
E73B 7A                 DATA  :7A         a4: ((LN2)^4)/4!
E73C 9D                 DATA  :9D         0.0096217389747
E73D A4                 DATA  :A4
E73E 81                 DATA  :81
                *
E73F 77                 DATA  :77         a5: ((LN2)^5)/5!
E740 AE                 DATA  :AE         0.0013337729375
E741 D1                 DATA  :D1
E742 FE                 DATA  :FE
                *
E743 00                 DATA  :00         End of table
E744 00                 DATA  :00
                *
                *******
                * LOG *
                *******
                *
                * MACC = LN (MACC).
                *
                * Method: Polynomial approximation.
                *
```

```
                        * 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
```



```
E795 33                 INX   SP        )
E796 33                 INX   SP        ) Suppress 4 bytes
E797 33                 INX   SP        ) on stack
E798 33                 INX   SP        )
E799 E1                 POP   H         HL=00E7
E79A CDDBE9             CALL  :E9DB     Copy v^2 into 00E7-EA
E79D D1                 POP   D         Get J in D
E79E 7A                 MOV   A,D       )
E79F 17                 RAL             )
E7A0 9F                 SBB   A         ) Convert J from 1 byte
E7A1 47                 MOV   B,A       ) into 4 byte into ABCD
E7A2 4F                 MOV   C,A       )
E7A3 CD26E1             CALL  :E126     Copy ABCD into MACC
E7A6 CDDEE3             CALL  :E3DE     MACC = INT(MACC)
E7A9 21B8E7             LXI   H,:E7B8   Addr ln(2)
E7AC CD59EA             CALL  :EA59     MACC=MACC*ln(2)  (=J*ln(2))
E7AF 21BCE7             LXI   H,:E7BC   Addr Taylor sum constants
E7B2 CDAAE5             CALL  :E5AA     Calc Taylor sum (= ln(X)
E7B5 C34DC1             JMP   :C14D     Popall, ret

                        * CONSTANTS FOR 'XLN':

E7B8 00         L1E298  DATA  :00       LN(2)
E7B9 B1                 DATA  :B1
E7BA 72                 DATA  :72
E7BB 18                 DATA  :18
                        *
E7BC 02         L1E299  DATA  :02       b1: FPT (2)
E7BD 80                 DATA  :80
E7BE 00                 DATA  :00
E7BF 00                 DATA  :00
                        *
E7C0 00                 DATA  :00       b3: about 2/3
E7C1 AA                 DATA  :AA       0.666666564181
E7C2 AA                 DATA  :AA
E7C3 A9                 DATA  :A9
                        *
E7C4 7F                 DATA  :7F       b5: about 2/5
E7C5 CC                 DATA  :CC       0.400018840613
E7C6 CF                 DATA  :CF
E7C7 45                 DATA  :45
                        *
E7C8 7F                 DATA  :7F       b7: about 2/7
E7C9 91                 DATA  :91       0.2845357266
E7CA AE                 DATA  :AE
E7CB AB                 DATA  :AB
                        *
E7CC 7E                 DATA  :7E       b9: about 2/9
E7CD 80                 DATA  :80       0.125
E7CE 00                 DATA  :00
E7CF 00                 DATA  :00
                        *
E7D0 00                 DATA  :00       End of table
E7D1 00                 DATA  :00
                        *
                        *
                        *
E7D2                    END

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

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



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

```
064 E801 2137E8            LXI   H,:E837   Addr FPT (0.25)
065 E804 E5                PUSH  H         Save pntr
066 E805 CD6DEA            CALL  :EA6D     MACC = MACC - 0.25
067 E808 CDEEE9            CALL  :E9EE     Take abs. value
068 E80B 213BE8            LXI   H,:E83B   Addr FPT (0.5)
069 E80E CD6DEA            CALL  :EA6D     MACC = MACC - 0.5
070 E811 CDEEE9            CALL  :E9EE     Take abs. value
071 E814 E1                POP   H         Get addr FPT (0.25)
072 E815 CD6DEA            CALL  :EA6D     MACC = MACC - 0.25
073 E818 21E300    L1E134  LXI   H,:00E3
074 E81B E5                PUSH  H
075 E81C CDD6E9            CALL  :E9D6     Copy MACC into 00E3-E6
076 E81F E3                XTHL            HL=00E3; stack: 00E7
077 E820 CD59EA            CALL  :EA59     MACC = 2 * MACC
078 E823 E1                POP   H         HL=00E7
079 E824 CDDBE9            CALL  :E9DB     Copy 2*MACC into 00E7-EA
080 E827 CD16EA            CALL  :EA16     Clear MACC + reg ABCD
081 E82A 213FE8            LXI   H,:E83F   Addr Taylor sum constants
082 E82D CDAAE5            CALL  :E5AA     Calc Taylor sum
083 E830 C34DC1            JMP   :C14D     Popall, ret
084
085                * CONSTANTS FOR 'XSIN' AND 'XCOS':
086
087 E833 01         FPHPI  DATA  :01       FPT (PI/2)
088 E834 C9                DATA  :C9
089 E835 0F                DATA  :0F
090 E836 DB                DATA  :DB
091                *
092 E837 7F         L1E304 DATA  :7F       FPT (0.25)
093 E838 80                DATA  :80
094 E839 00                DATA  :00
095 E83A 00                DATA  :00
096                *
097 E83B 00         L1E305 DATA  :00       FPT (0.5)
098 E83C 80                DATA  :80
099 E83D 00                DATA  :00
100 E83E 00                DATA  :00
101                *
102 E83F 03         L1E306 DATA  :03       a1: about PI*2
103 E840 C9                DATA  :C9       6.2831853
104 E841 0F                DATA  :0F
105 E842 DB                DATA  :DB
106                *
107 E843 86                DATA  :86       a2: about -(PI*2)^3/3!
108 E844 A5                DATA  :A5       -41.341681
109 E845 5D                DATA  :5D
110 E846 E2                DATA  :E2
111                *
112 E847 07                DATA  :07       a3: about (PI*2)^5/5!
113 E848 A3                DATA  :A3       81.602481
114 E849 34                DATA  :34
115 E84A 78                DATA  :78
116                *
117 E84B 87                DATA  :87       a4: about -(PI*2)^7/7!
118 E84C 99                DATA  :99       -76.581285
119 E84D 29                DATA  :29
120 E84E 9E                DATA  :9E
121                *
122 E84F 06                DATA  :06       a5: about (PI*2)^9/9!
123 E850 9F                DATA  :9F       39.760722
124 E851 0A                DATA  :0A
125 E852 FB                DATA  :FB
```



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

```
E880 F5          XALOG   PUSH  PSW
E881 C5                  PUSH  B
E882 D5                  PUSH  D
E883 E5                  PUSH  H
E884 2190E8              LXI   H,:E890    Addr 1/ln(10)
E887 CD20EA              CALL  :EA20      MACC = X*ln(10)
E88A CD67E6              CALL  :E667      MACC = e^(X*ln(10))
E88D C34DC1              JMP   :C14D      Popall, ret

                 * CONSTANT FOR 'XLOG' AND 'XALOG':

E890 7F          FLGTI   DATA  :7F        1/ln(10)
E891 DE                  DATA  :DE
E892 5B                  DATA  :5B
E893 D9                  DATA  :D9
                 *
                 *******
                 * TAN *
                 *******
                 *
                 * MACC = TAN (MACC)  (Angle in radians).
                 *
                 * Method: tan(X) = sin(X)/cos(X).
                 *         In-accurate for X close to 0 or close
                 *         to n*PI/2.
                 *
                 * Exit: All registers preserved.
                 *
E894 E5          XTAN    PUSH  H
E895 CD1EC2              CALL  :C21E      Save X on stack
E898 CDD9E7              CALL  :E7D9      MACC = cos(X)
E89B 21EF00              LXI   H,:00EF
E89E CD1CE1              CALL  :E11C      Store cos(X) in 00EF-F2
E8A1 CD34C2              CALL  :C234      Get X from stack
E8A4 CDD2E7              CALL  :E7D2      MACC = sin(X)
E8A7 CD08E1              CALL  :E108      MACC = sin(X)/cos(X)
E8AA E1                  POP   H
E8AB C9                  RET
                 *
                 ********
                 * ATAN *
                 ********
                 *
                 * MACC = ATAN (MACC)  (Angle expressed in radians).
                 *
                 * Method: Polynomial approximation.
                 *
                 * ATAN(Z) for -0.25 <= Z <= 0.25 approximated by:
                 *       F(X) = X*(1 - Q1*X^2 + Q2*X^4 - Q3*X^6).
                 *
                 * To cope with range:
                 *       ATAN(-Z) = - ATAN(Z).
                 *       ATAN(Z) = a(k) + ATAN((Z-b(k))/(Z*b(k)+1)),
                 *                 with k = 1, 2 or 3,
                 *                 a(k) = k*PI/7,
                 *                 b(k) = TAN(a(k))
                 *
                 * Values for k:
                 *       k=0 if ABS(Z) < 0.25
                 *       k=1 if 0.25 < ABS(Z) < 0.75
                 *       k=2 if 0.75 < ABS(Z) < 2
                 *       k=3 if ABS(Z) > 2.
```



```
                 *
                 * Then X = (Z-b(k))/(Z*b(k)+1), and
                 *      ATAN(Z) =  a(k) + F(X), if Z >= 0
                 *      ATAN(Z) = -a(k) - F(X), if Z < 0.
                 *
E8AC F5          XATAN   PUSH  PSW
E8AD C5                  PUSH  B
E8AE D5                  PUSH  D
E8AF E5                  PUSH  H
E8B0 CD71EB              CALL  :EB71      Check if Z=0
E8B3 CA43E9              JZ    :E943      Then abort
E8B6 F5                  PUSH  PSW        Save exp byte
E8B7 CDEEE9              CALL  :E9EE      reg ABCD = ABS(Z)
E8BA 21EF00              LXI   H,:00EF
E8BD CDDBE9              CALL  :E9DB      Copy ABS(Z) into 00EF-F2

                 * Calculate k:

E8C0 FE40                CPI   :40
E8C2 DAD3E8              JC    :E8D3      Jump if exp < #40
E8C5 FE7F                CPI   :7F
E8C7 3E01                MVI   A,:01
E8C9 CAE6E8              JZ    :E8E6      k=1 if exp=#7F
E8CC 215EC4              LXI   H,:C45E    Addr FPT(0)
E8CF E5                  PUSH  H
E8D0 C315E9              JMP   :E915      Cont with k=1, a(k)=0
E8D3 FE01        L1E141  CPI   :01
E8D5 3E02                MVI   A,:02
E8D7 CAE6E8              JZ    :E8E6      k=2 if exp=1
E8DA D2E3E8              JNC   :E8E3      k=3 if exp >1
E8DD 78                  MOV   A,B        Get hibyte mantissa
E8DE 07                  RLC
E8DF 07                  RLC
E8E0 3E01                MVI   A,:01      k=1 if (B)= 10...
                                          k=2 if (B)= 11...
E8E2 3F                  CMC
E8E3 3F          L1E142  CMC
E8E4 CE00                ACI   :00
                 *
E8E6 87          L1E143  ADD   A          Final k in A
E8E7 87                  ADD   A
E8E8 87                  ADD   A          *8
E8E9 213EE9              LXI   H,:E93E    Startaddr for a,b table
E8EC 5F                  MOV   E,A        )
E8ED 1600                MVI   D,:00      ) offset in DE
E8EF 19                  DAD   D
E8F0 E5                  PUSH  H          Addr a(k)
E8F1 110400              LXI   D,:0004
E8F4 19                  DAD   D
E8F5 E5                  PUSH  H          Addr b(k)
E8F6 CD59EA              CALL  :EA59      MACC = Z*b(k)
E8F9 2162C4              LXI   H,:C462    Addr FPT(1)
E8FC CD72EA              CALL  :EA72      MACC = Z*b(k)+1
E8FF 21DF00              LXI   H,:00DF
E902 CDDBE9              CALL  :E9DB      (Z*b(k)+1) into 00DF-E2
E905 21EF00              LXI   H,:00EF
E908 CDFBE9              CALL  :E9FB      ABS(Z) in MACC
E90B E1                  POP   H          Addr b(k)
E90C CD6DEA              CALL  :EA6D      MACC = Z-b(k)
E90F 21DF00              LXI   H,:00DF
E912 CD20EA              CALL  :EA20      MACC = X =
                                          = (Z-b(k))/(Z*b(k)+1)
```

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



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

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

                        * Error exit:

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



```
                        * From ATEST also used to copy MACC into ABCD.
                        * From L1E158 used to copy operand (pointed at
                        * by HL) into ABCD and into MACC.
                        *
E9EE 01007F     L1E155  LXI   B,:7F00   Set mask
E9F1 21D500     L1E156  LXI   H,:00D5   Addr MACC
E9F4 78                 MOV   A,B       Mask in A
E9F5 A6                 ANA   M         AND exp byte with mask
E9F6 A9                 XRA   C         Set sign bit = 0
E9F7 77                 MOV   M,A       Update exp byte MACC
                        *
E9F8 21D500     ATEST   LXI   H,:00D5   Addr MACC
E9FB CDF4EB     L1E158  CALL  :EBF4     Check if MEM = 0, get
                                        exp byte in A
E9FE CA16EA             JZ    :EA16     Then clear MACC + ABCD
EA01 5F                 MOV   E,A       exp byte in E
EA02 23                 INX   H         )
EA03 46                 MOV   B,M       )
EA04 23                 INX   H         ) Mantissa from MEM
EA05 4E                 MOV   C,M       ) into BCD
EA06 23                 INX   H         )
EA07 56                 MOV   D,M       )
EA08 21D500             LXI   H,:00D5   Addr MACC
EA0B C317EB             JMP   :EB17     Copy ABCD into MACC;
                                        exp from E in A, flags
                                        set on exp ORI 01
                        *
                        *
                        *
EA0E                    END
```

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

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

```
002                             ORG   :EA0E
003                     *
004                     *
005                     *
006                     ****************************************
007                     * COPY OPERAND INTO REGISTERS B,C,D,E *
008                     ****************************************
009                     *
010                     * Entry: HL points to operand.
011                     * Exit:  HL points to last byte of operand.
012                     *        AF preserved.
013                     *
014 EA0E 46             L1E159  MOV   B,M
015 EA0F 23                     INX   H
016 EA10 4E                     MOV   C,M
017 EA11 23                     INX   H
018 EA12 56                     MOV   D,M
019 EA13 23                     INX   H
020 EA14 5E                     MOV   E,M
021 EA15 C9                     RET
022                     *
023                     ************************************
024                     * CLEAR MACC AND REGISTERS A,B,C,D *
025                     ************************************
026                     *
027 EA16 21D500         AZERO   LXI   H,:00D5   Addr MACC
028 EA19 AF                     XRA   A         )
029 EA1A 47                     MOV   B,A       ) Clear ABCD
030 EA1B 4F                     MOV   C,A       )
031 EA1C 57                     MOV   D,A       )
032 EA1D C3DBE9                 JMP   :E9DB     Clear MACC
033                     *
034                     ***************************
035                     * FPT DIVIDE SUBROUTINE *
036                     ***************************
037                     *
038                     * MACC = MACC / MEM. Rounded quotient in MACC
039                     * and registers ABCD, exponent in E.
040                     *
041                     * Entry: HL points to operand.
042                     * Exit:  CY=1: Overflow, result invalid.
043                     *        CY=0: Result in ABCD, EHL corrupted.
044                     *
045 EA20 CDF4EB         ADIV    CALL  :EBF4     Test if MEM=0; exp byte in A
046 EA23 CA54EA                 JZ    :EA54     Then run divide by 0 error
047 EA26 F5                     PUSH  PSW       Save exp MEM
048 EA27 E680                   ANI   :80       Sign bit only
049 EA29 47                     MOV   B,A       Preserve sign
050 EA2A F1                     POP   PSW       Get exp MEM
051 EA2B E67F                   ANI   :7F       Skip sign bit
052 EA2D 2F                     CMA             ) 2-compl of exponent
053 EA2E 3C                     INR   A         )
054 EA2F FEC0                   CPI   :C0       Overflow in sign bit ?
055 EA31 CA46EA                 JZ    :EA46     Then run overflow error
056 EA34 E67F                   ANI   :7F       Only compl. exp MEM
057 EA36 B0                     ORA   B         Add sign
058 EA37 CD1DEB                 CALL  :EB1D     Subtract exponents
059 EA3A DA4BEA                 JC    :EA4B     Evt. run overflow error
060 EA3D CA16EA                 JZ    :EA16     If zero result: clear MACC +
061                                               ABCD
062 EA40 CD4AEC                 CALL  :EC4A     Run fixed division
063 EA43 D206EB                 JNC   :EB06     Round up if no overflow
```



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

```
                        *
                        * MACC = MACC - MEM.
                        *
                        * Entry: HL points to operand in memory.
                        * Exit:  CY=1: Overflow.
                        *        CY=0: Result in ABCD. EHL corrupted.
                        *
EA72 0600       AADD    MVI   B,:00     Zero mask
EA74 3E7F       AD10    MVI   A,:7F     Most possible value
EA76 32DE00             STA   :00DE     Set MACC >> MEM
EA79 CDF4EB             CALL  :EBF4     Test if MEM =0; exp in A
EA7C CAF8E9             JZ    :E9F8     Then clear MACC + ABCD
EA7F 78                 MOV   A,B       Get mask
EA80 AE                 XRA   M         XOR with exp (ADD: gives
                                        exp; SUB: gives -exp)
EA81 23                 INX   H
EA82 46                 MOV   B,M       )
EA83 23                 INX   H         )
EA84 4E                 MOV   C,M       ) Copy mantissa MEM into B
EA85 23                 INX   H         )
EA86 56                 MOV   D,M       )
EA87 5F                 MOV   E,A       Exp in E
EA88 21D500             LXI   H,:00D5   Addr MACC
EA8B 7E                 MOV   A,M       Get exp MACC
EA8C AB                 XRA   E         XOR with exp MEM
EA8D E680               ANI   :80       Sign only
EA8F 32D900             STA   :00D9     Store #80 if different signs
EA92 CDF4EB             CALL  :EBF4     Test if MACC=0; exp in A
EA95 CA11EB             JZ    :EB11     Jump if true
EA98 D5                 PUSH  D
EA99 7B                 MOV   A,E       Get exp MEM
EA9A CDE9C1             CALL  :C1E9     Sign extend
EA9D 5F                 MOV   E,A       Ext exp MEM in E
EA9E 7E                 MOV   A,M       Get exp MACC
EA9F CDE9C1             CALL  :C1E9     Sign extend
EAA2 93                 SUB   E         Calc difference
EAA3 D1                 POP   D
EAA4 32DE00             STA   :00DE     Save it
EAA7 FAB2EA             JM    :EAB2     If exp MACC < exp MEM:
                                        exchange ABCD and MACC
EAAA FE19               CPI   :19       Total bits in mantissa
EAAC DAC6EA             JC    :EAC6     OK if difference between
                                        both nrs <#19 in exp
EAAF C3F8E9             JMP   :E9F8     Else: Result is zero in
                                        MACC and ABCD

                        * Exchange MACC and ABCD:

EAB2 FEE7       L1E169  CPI   :E7       Total bits in mantissa
EAB4 DA16EB             JC    :EB16     If difference not too big
EAB7 73                 MOV   M,E       Ext exp MEM in MACC
EAB8 2F                 CMA             )
EAB9 3C                 INR   A         ) A = ext exp old MACC
EABA 23                 INX   H
EABB 5E                 MOV   E,M       ) Exchange 1st byte MACC
EABC 70                 MOV   M,B       ) mantissa and byte in B
EABD 43                 MOV   B,E       )
EABE 23                 INX   H
EABF 5E                 MOV   E,M       ) Exchange 2nd byte MACC
EAC0 71                 MOV   M,C       ) mantissa and byte in C
EAC1 4B                 MOV   C,E       )
EAC2 23                 INX   H
```



```
EAC3 5E                 MOV   E,M       ) Exchange 3rd byte MACC
EAC4 72                 MOV   M,D       ) mantissa and byte in D
EAC5 53                 MOV   D,E       )
                                        Now orig MACC in ABCD and
                                        orig MEM in MACC
EAC6 1E00       L1E170  MVI   E,:00
EAC8 CD55EB             CALL  :EB55     Shift BCDE right A places
EACB 3AD900             LDA   :00D9     Get result XOR sign bits
EACE B7                 ORA   A
EACF 21D800             LXI   H,:00D8   Addr lobyte MACC
EAD2 FAEFEA             JM    :EAEF     Jump if different signbits

                        * If both signs equal:

EAD5 7E                 MOV   A,M       )
EAD6 82                 ADD   D         )
EAD7 57                 MOV   D,A       )
EAD8 2B                 DCX   H         )
EAD9 7E                 MOV   A,M       ) Add mantissa MACC to BCD
EADA 89                 ADC   C         ) Result in BCD.
EADB 4F                 MOV   C,A       )
EADC 2B                 DCX   H         )
EADD 7E                 MOV   A,M       )
EADE 88                 ADC   B         )
EADF 47                 MOV   B,A       )
EAE0 D206EB             JNC   :EB06     Jump if no overflow
EAE3 CD70EB             CALL  :EB70     Else: shift BCDE right 1 bit
EAE6 CDD9EB             CALL  :EBD9     Incr exponent
EAE9 DA46EA             JC    :EA46     Evt run overflow error
EAEC C306EB             JMP   :EB06     Round up

                        * If both signs not equal:

EAEF AF         L1E171  XRA   A         )
EAF0 93                 SUB   E         ) Compl exp in E
EAF1 5F                 MOV   E,A       )
EAF2 7E                 MOV   A,M       )
EAF3 9A                 SBB   D         )
EAF4 57                 MOV   D,A       ) Subtract BCD from mantissa
EAF5 2B                 DCX   H         ) MACC. Result in BCD.
EAF6 7E                 MOV   A,M       )
EAF7 99                 SBB   C         )
EAF8 4F                 MOV   C,A       )
EAF9 2B                 DCX   H         )
EAFA 7E                 MOV   A,M       )
EAFB 98                 SBB   B         )
EAFC 47                 MOV   B,A       )
EAFD DC7DEB             CC    :EB7D     Correct if overflow
EB00 F496EB     AD10A   CP    :EB96     Evt normalize BCDE
EB03 F216EA             JP    :EA16     and clear MACC + ABCD

                        * Normal exit:

EB06 CDC3EB     ADD11   CALL  :EBC3     Round up BCD, result in MACC
                                        exp in E
EB09 DA46EA             JC    :EA46     Evt run overflow error
EB0C 7B         L1E174  MOV   A,E       Get exponent
EB0D F601               ORI   :01       Set flags on exp OR 1
EB0F 7B                 MOV   A,E       Exp in A
EB10 C9                 RET

                        * If operand = 0:
```

```
                        *
                        **********************
                        * FPT: ADD EXPONENTS *
                        **********************
                        *
                        * Adds the exponent of the MACC to the exponent
                        * of a operand in memory.
                        *
                        * Entry: HL points to FPT number in memory.
                        *        A contains its exponent.
                        *        Other number in MACC.
                        * Exit:    Z=1:       MACC=0; HL=00D5
                        *        CY=1:        Overflow: HL=00D5; MACC pres.
                        *                     A: Sum of signed exponents SHL 1
                        *        Z=0, CY=0: O.K.: HL=00D5; sum of exponents
                        *                     in MACC.
                        *
EB1D 47         MDEX    MOV   B,A       Exp MEM in B
EB1E 23                 INX   H
EB1F 4E                 MOV   C,M       )
EB20 23                 INX   H         ) Copy mantissa MEM in CDE
EB21 56                 MOV   D,M       )
EB22 23                 INX   H         )
EB23 5E                 MOV   E,M       )
EB24 CDF1EB             CALL  :EBF1     Test MACC=0; Exp MACC in A
EB27 C8                 RZ              Abort if MACC=0, Z=1
EB28 78                 MOV   A,B       Get exp MEM in A
EB29 CDE9C1             CALL  :C1E9     Sign extend
EB2C CDBAC1             CALL  :C1BA     Add exponents, result in MAC
EB2F D8                 RC              Abort if overflow, CY=1
EB30 78                 MOV   A,B       Get orig exp MEM
EB31 E680               ANI   :80       sign bit only
EB33 AE                 XRA   M         Evt correct sign
EB34 77                 MOV   M,A       Exp back into MACC
EB35 3E01               MVI   A,:01
EB37 B7                 ORA   A
EB38 C9                 RET
                        *
                        ********************************
                        * SHIFT BCDE LEFT (A) POSITIONS *
                        ********************************
                        *
                        * Exit: AF preserved.
                        *
EB39 F5         LSHN    PUSH  PSW
EB3A 6F                 MOV   L,A       Nr of shifts in L
EB3B 2D         L1E180  DCR   L
EB3C FA46EB             JM    :EB46     Abort if ready
EB3F B7                 ORA   A         Clear CY
EB40 CD48EB             CALL  :EB48     Shift BCDE left 1 position
EB43 C33BEB             JMP   :EB3B     Next shift
EB46 F1         L1E182  POP   PSW
EB47 C9                 RET
                        *
                        *
```



```
                        **********************
                        * MULTIPLY BCDE * 2 *
                        **********************
                        *
                        * Shifts BCDE left 1 position. Entry CY goes to
                        * lsb of E.
                        *
                        * Exit: A corrupted, HL preserved.
                        *       F set on result.
                        *
EB48 7B         L1E183  MOV   A,E
EB49 17                 RAL             Shift left E
EB4A 5F                 MOV   E,A
EB4B 7A                 MOV   A,D
EB4C 17                 RAL             Shift left D
EB4D 57                 MOV   D,A
EB4E 79                 MOV   A,C
EB4F 17                 RAL             Shift left C
EB50 4F                 MOV   C,A
EB51 78                 MOV   A,B
EB52 8F                 ADC   A         B=2*B+CY
EB53 47                 MOV   B,A
EB54 C9                 RET
                        *
                        ************************************
                        * SHIFT BCDE RIGHT (A) POSITIONS *
                        ************************************
                        *
EB55 2E08       RSHN    MVI   L,:0008   Nr of shifts for 1 byte
EB57 BD         L1E185  CMP   L
EB58 FA64EB             JM    :EB64     Jump if A<8
                        *
                        * Shift 8 bits right:
                        *
EB5B 5A                 MOV   E,D       )
EB5C 51                 MOV   D,C       ) Shift 8 pos in one
EB5D 48                 MOV   C,B       ) time
EB5E 0600               MVI   B,:00     )
EB60 95                 SUB   L         Update nr of shifts left
EB61 C257EB             JNZ   :EB57     Again if not ready
                        *
                        * Shift 1 bit:
                        *
EB64 B7         L1E186  ORA   A
EB65 C8                 RZ              Abort if ready
EB66 6F                 MOV   L,A       L is nr of shifts
EB67 B7         L1E187  ORA   A
EB68 CD70EB             CALL  :EB70     Shift BCDE right one bit
EB6B 2D                 DCR   L         Update shift count
EB6C C267EB             JNZ   :EB67     Again if not ready
EB6F C9                 RET
                        *
                        ********************
                        * DIVIDE BCDE BY 2 *
                        ********************
                        *
                        * Shifts contents BCDE right 1 position.
                        *
                        * Exit: AF corrupted, HL preserved.
                        *
EB70 78         L1E188  MOV   A,B
EB71 1F                 RAR             Shift right B
```

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



```
436                ******************************
437                * NORMALIZE CONTENTS B,C,D,E *
438                ******************************
439                *
440                * Shifts contents BCDE left until the msb = 1.
441                *
442                * Exit: A: Minus number of shifts.
443                *       HL restored, S+Z-flag set on result.
444                *       CY=1: BCDE was zero.
445                *
446 EBA0 E5        L1E192  PUSH  H
447 EBA1 2E20              MVI   L,:20      Max 32 bits to shift
448 EBA3 78        L1E193  MOV   A,B        Get 1st byte
449 EBA4 B7                ORA   A
450 EBA5 C2BAEB            JNZ   :EBBA      If '1'-bits in it
451
452                * Shift 8 bits at once:
453
454 EBA8 41                MOV   B,C        )
455 EBA9 4A                MOV   C,D        ) Shift 1 byte
456 EBAA 53                MOV   D,E        )
457 EBAB 5F                MOV   E,A        )
458 EBAC 7D                MOV   A,L
459 EBAD D608              SUI   :08        Count minus 8 bits
460 EBAF 6F                MOV   L,A
461 EBB0 C2A3EB            JNZ   :EBA3      Continu if not ready
462 EBB3 E1                POP   H
463 EBB4 37                STC              If 4* 8 bits shifted and no
464                                         '1' found: BCDE was 0: CY=1
465 EBB5 C9                RET
466
467                * Shift 1 bit:
468
469 EBB6 2D        L1E194  DCR   L          Update count
470 EBB7 CD48EB            CALL  :EB48      Shift BCDE 1 bit left
471 EBBA F2B6EB    L1E195  JP    :EBB6      Again if msb <> 0
472
473                * If ready:
474
475 EBBD 7D                MOV   A,L        Get nr of shifts left
476 EBBE D620              SUI   :20        Calc neg nr of shifts done
477 EBC0 B7                ORA   A
478 EBC1 E1                POP   H
479 EBC2 C9                RET
480                *
481                *********
482                * ROUND *
483                *********
484                *
485                * Rounds up a FPT mantissa in BCD(E). Result in
486                * MACC, exponent also in E.
487                *
488                * Entry: FPT mantissa in BCDE.
489                * Exit:  CY=1: overflow.
490                *        All registers corrupted.
491                *
492 EBC3 7B        L1E196  MOV   A,E        Get lobyte mantissa
493 EBC4 B7                ORA   A
494 EBC5 FCD1EB            CM    :EBD1      Round up BCD if (E)
495                                         >= #80
496 EBC8 D8                RC               Abort if overflow
497 EBC9 21D500            LXI   H,:00D5    Addr MACC
```

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



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

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

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

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



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

```
126 EC7D 6B                 MOV   L,E       )
127 EC7E 62                 MOV   H,D       ) Remainder in EHL
128 EC7F 5F                 MOV   E,A       )
129 EC80 1601               MVI   D,:01
130 EC82 48                 MOV   C,B
131 EC83 C5         L1E206  PUSH  B
132 EC84 44                 MOV   B,H
133 EC85 4D                 MOV   C,L
134 EC86 21DA00             LXI   H,:00DA
135 EC89 AF                 XRA   A
136 EC8A 96                 SUB   M
137 EC8B 23                 INX   H
138 EC8C 79                 MOV   A,C
139 EC8D 9E                 SBB   M
140 EC8E 4F                 MOV   C,A
141 EC8F 23                 INX   H
142 EC90 78                 MOV   A,B
143 EC91 9E                 SBB   M
144 EC92 47                 MOV   B,A
145 EC93 23                 INX   H
146 EC94 7B                 MOV   A,E
147 EC95 9E                 SBB   M
148 EC96 5F                 MOV   E,A
149 EC97 69                 MOV   L,C
150 EC98 60                 MOV   H,B
151 EC99 C1                 POP   B
152 EC9A 3ADA00     L1E207  LDA   :00DA
153 EC9D 07                 RLC
154 EC9E 78                 MOV   A,B
155 EC9F 17                 RAL
156 ECA0 3F                 CMC
157 ECA1 D0                 RNC
158 ECA2 1F                 RAR
159 ECA3 7D                 MOV   A,L
160 ECA4 17                 RAL
161 ECA5 6F                 MOV   L,A
162 ECA6 7C                 MOV   A,H
163 ECA7 17                 RAL
164 ECA8 67                 MOV   H,A
165 ECA9 CD48EB             CALL  :EB48     Shift BCDE left 1 bit
166 ECAC 7A                 MOV   A,D
167 ECAD 0F                 RRC
168 ECAE DA83EC             JC    :EC83
169 ECB1 C5         L1E208  PUSH  B
170 ECB2 44                 MOV   B,H
171 ECB3 4D                 MOV   C,L
172 ECB4 2ADB00             LHLD  :00DB
173 ECB7 09                 DAD   B
174 ECB8 3ADD00             LDA   :00DD
175 ECBB 8B                 ADC   E
176 ECBC 5F                 MOV   E,A
177 ECBD C1                 POP   B
178 ECBE 3ADA00             LDA   :00DA
179 ECC1 C39DEC             JMP   :EC9D
180 ECC4 6B         L1E209  MOV   L,E
181 ECC5 62                 MOV   H,D
182 ECC6 5F                 MOV   E,A
183 ECC7 50                 MOV   D,B
184 ECC8 48                 MOV   C,B
185 ECC9 C3B1EC             JMP   :ECB1
186                 *
187                 *
```



```
188                 ************************************
189                 * AMD: ISSUE COMMAND TO MATH.CHIP *
190                 ************************************
191                 *
192                 * Entry: HL points to command.
193                 * Exit:  HL updated, A corrupted, BCDEF preserved.
194                 *
195 ECCC 7E        MPT15   MOV   A,M       Get command
196 ECCD 23                INX   H
197 ECCE 3202FB            STA   :FB02     Issue cmd to math.chip
198 ECD1 C9                RET
199                 *
200                 ******************************
201                 * AMD: TURN OFF ERROR STATUS *
202                 ******************************
203                 *
204                 * Exit: All registers preserved.
205                 *
206 ECD2 F5        MPT16   PUSH  PSW
207 ECD3 AF                XRA   A
208 ECD4 3202FB            STA   :FB02     Cmd math.chip = 0
209 ECD7 F1                POP   PSW
210 ECD8 C9                RET
211                 *
212                 ****************************************
213                 * AMD: LOAD 16-BIT DATA INTO MATH.CHIP *
214                 ****************************************
215                 *
216                 * Entry: 1st byte in A, 2nd on stack.
217                 *
218 ECD9 3200FB      MPT17   STA   :FB00     Load 1st byte in math.chip
219 ECDC F1                POP   PSW
220 ECDD 3200FB            STA   :FB00     Load data in math.chip
221 ECE0 C9                RET
222                 *
223                 **************************
224                 * part of 'IAND' (1E32C) *
225                 **************************
226                 *
227 ECE1 CD8CE3     L1E213  CALL  :E38C     Copy MACC into EBCA
228 ECE4 CD35E3             CALL  :E335     Run IAND
229 ECE7 C385E3             JMP   :E385     Copy ABCD into MACC
230                 *
231                 **************************
232                 * part of 'IOR' (1E345) *
233                 **************************
234                 *
235 ECEA CD8CE3     L1E214  CALL  :E38C     Copy MACC into EBCA
236 ECED CD4CE3             CALL  :E34C     Run IOR
237 ECF0 C385E3             JMP   :E385     Copy ABCD into MACC
238                 *
239                 **************************
240                 * part of 'IXOR' (1E35C) *
241                 **************************
242                 *
243 ECF3 CD8CE3     L1E215  CALL  :E38C     Copy MACC into EBCA
244 ECF6 CD63E3             CALL  :E363     Run IXOR
245 ECF9 C385E3             JMP   :E385     Copy ABCD into MACC
246                 *
247                 ********************************************
248                 * COPY MACC INTO REGISTERS A,B,C,D; CMA *
249                 ********************************************
```

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



```
ED19 F5         ZIAND   PUSH  PSW
ED1A C5                 PUSH  B
ED1B D5                 PUSH  D
ED1C E5                 PUSH  H
ED1D CD4FED             CALL  :ED4F     Copy MTOS into EBCA
ED20 CD35E3             CALL  :E335     Run IAND
ED23 C33DED             JMP   :ED3D     Result into MTOS
                        *
                        *************
                        * AMD: IOR *
                        *************
                        *
                        * MTOS = MTOS IOR MEM.
                        *
                        * Entry: HL points to operand in memory.
                        * Exit:  All registers preserved.
                        *
ED26 F5         ZIOR    PUSH  PSW
ED27 C5                 PUSH  B
ED28 D5                 PUSH  D
ED29 E5                 PUSH  H
ED2A CD4FED             CALL  :ED4F     Copy MTOS into EBCA
ED2D CD4CE3             CALL  :E34C     Run IOR
ED30 C33DED             JMP   :ED3D     Result into MTOS
                        *
                        *************
                        * AMD: IXOR *
                        *************
                        *
                        * MTOS = MTOS IXOR MEM.
                        *
                        * Entry: HL points to operand in memory.
                        * Exit:  All registers preserved.
                        *
ED33 F5         ZIXOR   PUSH  PSW
ED34 C5                 PUSH  B
ED35 D5                 PUSH  D
ED36 E5                 PUSH  H
ED37 CD4FED             CALL  :ED4F     Copy MTOS into EBCA
ED3A CD63E3             CALL  :E363     Run IXOR; result in ABCD
ED3D CD5FE5     ZORT1   CALL  :E55F     Copy ABCD into MTOS
ED40 C34DC1             JMP   :C14D     Popall, ret
                        *
                        *************
                        * AMD: INOT *
                        *************
                        *
                        * MTOS = INOT (MTOS).
                        *
                        * Exit: All registers preserved.
                        *
                        * REMARK: Wrong routine: MTOS is made -MTOS,
                        *         and then 1 is added. So result is
                        *         INOT (MTOS)+2.
                        *         Correct would be: Add -1.
                        *
ED43 CD22E5     ZINOT   CALL  :E522     Change sign MTOS (INT)
ED46 E5                 PUSH  H
ED47 2120C4             LXI   H,:C420   Addr INT (1)
ED4A CDF4E4             CALL  :E4F4     MTOS = - MTOS + 1
ED4D E1                 POP   H
ED4E C9                 RET
```

```
                        *
                        ****************************************
                        * AMD: COPY MTOS INTO REGISTERS E,B,C,A *
                        ****************************************
                        *
ED4F CD6FE5     ZIGTP   CALL  :E56F     Copy MTOS into ABCD
ED52 C304ED             JMP   :ED04     Copy ABCD into EBCA
                        *
                        ************
                        * AMD: SHR *
                        ************
                        *
                        * Shifts MTOS right MEM positions.
                        *
                        * Entry: HL points to INT number in memory.
                        * Exit:  All registers preserved.
                        *
ED55 F5         ZSHR    PUSH  PSW
ED56 C5                 PUSH  B
ED57 D5                 PUSH  D
ED58 E5                 PUSH  H
ED59 CDB2E3             CALL  :E3B2     Check value of MEM. Value in
                                        A. Clear ABCDE if too big
ED5C CC7CED             CZ    :ED7C     Else: Copy MTOS in BCDE
ED5F CD55EB             CALL  :EB55     Shift BCDE right A positions
ED62 78         ZRREG   MOV   A,B       )
ED63 41                 MOV   B,C       )
ED64 4A                 MOV   C,D       ) Copy BCDE into ABCD
ED65 53                 MOV   D,E       )
ED66 CD5FE5             CALL  :E55F     Copy ABCD into MTOS
ED69 C34DC1             JMP   :C14D     Popall, ret
                        *
                        ************
                        * AMD: SHL *
                        ************
                        *
                        * Shifts MTOS left MEM positions.
                        *
                        * Entry: HL points to INT number in memory.
                        * Exit:  All registers preserved.
                        *
ED6C F5         ZSHL    PUSH  PSW
ED6D C5                 PUSH  B
ED6E D5                 PUSH  D
ED6F E5                 PUSH  H
ED70 CDB2E3             CALL  :E3B2     Test value of MEM. Value in
                                        A. Clear ABCDE if too big
ED73 CC7CED             CZ    :ED7C     If OK: Copy MTOS into BCDE
ED76 CD39EB             CALL  :EB39     Shift BCDE left A positions
ED79 C362ED             JMP   :ED62     Copy BCDE into MTOS
                        *
                        ****************************************
                        * AMD: COPY MTOS INTO REGISTERS B,C,D,E *
                        ****************************************
                        *
                        * Exit: AHL preserved.
                        *
ED7C F5         ZGBCDE  PUSH  PSW
ED7D CD6FE5             CALL  :E56F     Copy MTOS into ABCD
ED80 C313ED             JMP   :ED13     Copy ABCD into BCDE
                        *
                        *
```



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

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



```
EDC4 2AA302             LHLD  :02A3     Get startaddr free RAM spac
EDC7 E5                 PUSH  H         and save it on stack
EDC8 EB                 XCHG            and in DE; HL is array pntr
EDC9 78                 MOV   A,B       )
EDCA CDFDED             CALL  :EDFD     ) Save length array in 1st
EDCD 79                 MOV   A,C       ) location free RAM
EDCE CDFDED             CALL  :EDFD     )
EDD1 78         L1E240  MOV   A,B
EDD2 B1                 ORA   C
EDD3 CAE5ED             JZ    :EDE5     Abort if ready
EDD6 7E                 MOV   A,M
EDD7 23                 INX   H
EDD8 E5                 PUSH  H         Addr array pntr on stack
EDD9 66                 MOV   H,M       ) Addr string element in HL
EDDA 6F                 MOV   L,A       )
EDDB CDEDED             CALL  :EDED     Store element in free RAM
EDDE E1                 POP   H         Get array pntr back
EDDF 23                 INX   H         Pnts to next element
EDE0 0B                 DCX   B         ) Decr length still to be
EDE1 0B                 DCX   B         ) done
EDE2 C3D1ED             JMP   :EDD1     Continu
                        *
                        * If ready:
                        *
EDE5 E1         L1E241  POP   H         Get startaddr new string
EDE6 EB                 XCHG            in DE; HL is end used area
EDE7 CD1ADE             CALL  :DE1A     Calc length of string
EDEA EB                 XCHG            in DE
EDEB C1                 POP   B
EDEC C9                 RET
                        *
                        * COPY STRING ELEMENT INTO FREE RAM:
                        *
EDED C5         L1E242  PUSH  B
EDEE 46                 MOV   B,M       Get length of string in B
EDEF 7E         L1E243  MOV   A,M       Byte in A
EDF0 23                 INX   H         Pnts to next byte
EDF1 CDFDED             CALL  :EDFD     Copy byte into free RAM
EDF4 78                 MOV   A,B       )
EDF5 D601               SUI   :01       ) Update length
EDF7 47                 MOV   B,A       )
EDF8 D2EFED             JNC   :EDEF     Next byte if not ready
EDFB C1                 POP   B
EDFC C9                 RET
                        *
                        * STORE STRINGDATA IN FREE RAM SPACE:
                        *
                        * Moves 1 byte of a string array element into the
                        * free RAM space and checks for 'OUT OF MEMORY'.
                        *
                        * Entry: DE: Points to 1st free address in RAM.
                        *        A:  Byte to be moved.
                        * Exit:  DE updated, BCHL preserved.
                        *
EDFD 12         L1E244  STAX  D         Store byte in free RAM
EDFE 13                 INX   D         Update RAM pntr
EDFF E5                 PUSH  H
EE00 2AA502             LHLD  :02A5     Get addr bottom screen RAM
EE03 CD14DE             CALL  :DE14     End free RAM reached ?
EE06 DA10DA             JC    :DA10     Then run error 'OUT OF
                                        MEMORY'
EE09 E1                 POP   H
```

```
EE0A C9                         RET
                    *
                    *
                    *
EE0B                            END
```

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

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


```
                            ORG    :EE0B
                    *
                    *
                    *
                    **************
                    * (not used) *
                    **************
                    *
EE0B E5             L1E245  PUSH   H
EE0C CD91D8                 CALL   :D891
                    *
                    ***********************************************
                    * LOADA: READ ARRAY DATA FROM TAPE INTO ARRAY *
                    ***********************************************
                    *
                    * Reads block(s) from tape into the free RAM space
                    * and move it afterwards into the array.
                    *
                    * Entry: HL: Length of name requested.
                    *        A:  Variable type.
                    *        On stack: Address where to dump data.
                    *
EE0F C5             L1E273  PUSH   B
EE10 F5                     PUSH   PSW          Save var.type
EE11 010032                 LXI    B,:3200      File type in B, C for load
                                                during program
EE14 CD81D6                 CALL   :D681        Open read file
EE17 E1                     POP    H            Get var.type in H
EE18 CDB5EF                 CALL   :EFB5        Read var.type from tape
EE1B BC                     CMP    H            Correct type ?
EE1C C21ADA                 JNZ    :DA1A        Run error 'TYPE MISMATCH'
                                                if not
EE1F FE20                   CPI    :20
EE21 CA38EE                 JZ     :EE38        Jump if string arrays

                    * If INT/FPT arrays:

EE24 E1                     POP    H
EE25 E3                     XTHL                Get dumpaddr in HL
EE26 EB                     XCHG                Get length in HL
EE27 19                     DAD    D            HL is end dump area
EE28 EB                     XCHG                in DE; HL is start dump
EE29 CDD102                 CALL   :02D1        Read block from tape
EE2C D2B3D2                 JNC    :D2B3        Evt run 'LOADING ERROR ..'
EE2F CDD402         L1E246  CALL   :02D4        Stop reading
EE32 3A4000                 LDA    :0040        Get POROM
EE35 C3A6EF                 JMP    :EFA6        Select ROM bank 0, abort

                    * If string arrays:

EE38 D5             L1E247  PUSH   D
EE39 2AA502                 LHLD   :02A5        Get bottom screen RAM
EE3C EB                     XCHG                in DE
EE3D 2AA302                 LHLD   :02A3        Get begin free RAM space
EE40 E5                     PUSH   H
EE41 CD97D8                 CALL   :D897        Read block from tape into
                                                free RAM area; evt.run error
EE44 E1                     POP    H            Get begin free RAM
EE45 56                     MOV    D,M          )
EE46 23                     INX    H            ) Length of array in DE
EE47 5E                     MOV    E,M          )
EE48 23                     INX    H
```

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



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

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



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

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



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

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

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

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