```
                        ORG   :E000
                *
                *
                *
                *   ============================
                *** ENCODING / UTILITY PACKAGES ***
                *   ============================
                *
                * Called by RST 1; DATA XX. XX indicates the
                * offset of E000 for the different entrypoints.
                *
                * Contains also the key encoding routines and the
                * Heap organisation routines.
                *
                ***************
                * ENTRYPOINTS *
                ***************
                *
E000 C324E0     ELINE   JMP   :E024     Encode a BASIC line
E003 C32AE7     ELN     JMP   :E72A     Encode a linenr
E006 C345E1     ETCON   JMP   :E145     Encode a constant
E009 C374EA     USTART  JMP   :EA74     Start Utility package
E00C C390EF     DBOOT   JMP   :EF90     Disc bootstrap
E00F C39CE9     MHREQ   JMP   :E99C     Heap request
E012 C33FE9     MINKEY  JMP   :E93F     Encode a key input
E015 C335E9     L3E394  JMP   :E935     Get inputs from keyb
                                        or DINC
                *
                *   ================
                *** ENCODING PACKAGE ***
                *   ================
                *
                * Generally in the encoding routines, HL points to
                * the first free location in the EBUF. C points to
                * the input text line.
                *
                ************************
                * UPDATE EBUF POINTER *
                ************************
                *
                * Increments the input pointer and checks if the
                * encoded input buffer (EBUF) is full.
                *
                * Entry: HL: input pointer EBUF.
                * Exit:  HL: updated pointer.
                *        Other registers preserved.
                *
E018 23         INXCH   INX   H         Incr pointer
E019 F5                 PUSH  PSW
E01A 7D                 MOV   A,L       Get lobyte in A
E01B FEBE               CPI   :BE       Max value reached?
E01D 3E1A               MVI   A,:1A     Buffer full ?
E01F D2F5D9             JNC   :D9F5     Then run error 'LINE TOO
                                        COMPLEX'
E022 F1                 POP   PSW
E023 C9                 RET
                *
                ***************************
                * ENCODE A BASIC COMMAND *
                ***************************
                *
                * Looks for a match between input on line and the
```



```
                * BASIC command table #CBBF. Error exit 'COMMAND
                * INVALID' if the 2 high order bits of the code in
                * the table don't match the mask in D. Else the
                * code (#80 + low order 6 bits from code in table)
                * are stored in EBUF.
                *
                * Entry: HL: 1st free position in EBUF.
                *        D : Mask for direct command (#80) or
                *            program input (#40).
                *        C : Position on current line.
                * Exit:  HL: Updated.
                *        C : Points to 1st not used input in line.
                *        E : Code just stored.
                *        BD preserved, AF corrupted.
                *        Exit with jump to address found in table.
                *        #E04F is on stack as next returnaddress,
                *        entry DE is next on stack.
                *
E024 D5         L3E2    PUSH  D
E025 E5                 PUSH  H
E026 214FE0             LXI   H,:E04F   Returnaddr after finishing
                                        encoding
E029 E3                 XTHL            Save it on stack
E02A E5                 PUSH  H         Save HL again
E02B 1E02               MVI   E,:02     Addit. offset for finding
                                        instruction in table
E02D 21BFCB             LXI   H,:CBBF   Startaddr table with strings
                                        Basiccommands
E030 D5                 PUSH  D
E031 CD34CA             CALL  :CA34     Find instr in table
E034 D1                 POP   D         Get mask for direct/program
E035 7E                 MOV   A,M       Get instr code from table
E036 E6C0               ANI   :C0       ) Check if it
E038 A2                 ANA   D         ) is a valid command
E039 3E18               MVI   A,:18     If not: Run error
E03B CAF5D9             JZ    :D9F5     'COMMAND INVALID'
E03E 7E                 MOV   A,M       Get instr code from table
E03F 23                 INX   H
E040 E63F               ANI   :3F       ) Make it a token
E042 F680               ORI   :80       )
E044 5F                 MOV   E,A       Store token in E
E045 7E                 MOV   A,M       )
E046 23                 INX   H         ) Get addr of encoding
E047 66                 MOV   H,M       ) instruction in HL
E048 6F                 MOV   L,A       )
E049 E3                 XTHL            and save it on stack
E04A 73                 MOV   M,E       Store token in EBUF
E04B CD18E0             CALL  :E018     Update EBUF pointer
E04E C9                 RET             Go to encoding instruction
                                        return afterwards to E04F
                *
                * End of encoding of an input line (after running
                * the command specific processing).
                * Checks if character after BASIC command is CR or
                * ':'.
                *
                * Entry: C : points to position on current line.
                *        On stack: Original DE.
                *
E04F D1         L3E3    POP   D         Get type of command in D
E050 CDD2DD             CALL  :DDD2     Get char from line, neglect
                                        tab + space
```

```
E053 0C              INR   C         Pntr to next char
E054 FE3A            CPI   :3A       ':' ?
E056 CA24E0          JZ    :E024     Then encode next instr
E059 FE0D            CPI   :0D       'CR' ?
E05B C20BDA          JNZ   :DA0B     Run 'SYNTAX ERROR' if not
E05E C9              RET
                     *
                     ******************************
                     * ENCODE 'FOR - TO - (STEP)' *
                     ******************************
                     *
                     * Valid for all specific encoding routines:
                     * HL points to 1st free EBUF location.
                     *
E05F CDFEE0     EFOR     CALL  :E0FE     Encode 'LET'
E062 FE20                CPI   :20       String type ?
E064 CA1ADA              JZ    :DA1A     Then run error 'TYPE
                                         MISMATCH'
E067 FE10                CPI   :10       INT type ?
E069 1188E0              LXI   D,:E088   Addr 'TO' table
E06C C39AE0              JMP   :E09A     Get addr encoding instr and
                                         go to it

                     * Encode 'TO':

E06F CC6DE3     L3E5     CZ    :E36D     Encode a INT expr
E072 C476E3              CNZ   :E376     Encode a FPT expr
E075 1190E0              LXI   D,:E090   Addr 'STEP' table
E078 C39AE0              JMP   :E09A     Get addr encoding instr and
                                         go to it

                     * Encode 'STEP':

E07B CC6DE3     L3E6     CZ    :E36D     Encode a INT expr
E07E C476E3              CNZ   :E376     Encode a FPT expr
E081 C9                  RET

                     * End encoding:

E082 36FF       L3E363   MVI   M,:FF     FF in EBUF as separator
E084 CD18E0              CALL  :E018     Update EBUF pointer
E087 C9                  RET

                     * Table:

E088 02         L3E395   DATA  :02
E089 54                  DATA  :54       T
E08A 4F                  DATA  :4F       O
E08B 6FE0                DBL   :E06F     Addr encode 'TO'
                     *
E08D 00                  DATA  :00       'TO' not found:
E08E 0BDA                DBL   :DA0B     Run 'SYNTAX ERROR'
                     *
E090 04         L3E396   DATA  :04
E091 53                  DATA  :53       S
E092 54                  DATA  :54       T
E093 45                  DATA  :45       E
E094 50                  DATA  :50       P
E095 7BE0                DBL   :E07B     Addr encode 'STEP'
                     *
E097 00                  DATA  :00       'STEP' not found:
E098 82E0                DBL   :E082     End command
```



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

                     * If NEXT <variable>:

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

                     * Encode 'THEN':

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

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



```
312                 * if type non-existing).
313                 *
314                 * Exit: DE: Offset of left-side variable.
315                 *       A and B: Left-side type.
316                 *       C, HL updated. F corrupted.
317                 *
318 E0FE CDBCE5      ELET    CALL  :E5BC     Encode var or array ref
319 E101 CD67E8              CALL  :E867     Check next char is '='
320 E104 3D                  DATA  :3D
321 E105 3A3601              LDA   :0136     Get type latest expression
322 E108 FE00                CPI   :00       FPT ?
323 E10A CA76E3              JZ    :E376     Then encode FPT expr
324 E10D FE10                CPI   :10       INT ?
325 E10F CA6DE3              JZ    :E36D     Then encode INT expr
326 E112 C3A1E3      .       JMP   :E3A1     Else encode STR expr
327                 *
328                 ****************************************
329                 * ENCODE 'INPUT' AND 'INPUT <STRING>' *
330                 ****************************************
331                 *
332 E115 CDD2DD      EINPUT  CALL  :DDD2     Get char from line, neglect
333                                           tab + space
334 E118 FE22                CPI   :22       " ?
335 E11A C227E1              JNZ   :E127     Encode 'READ' if no string
336                                           in INPUT statement
337
338                 * If 'INPUT <string>':
339
340 E11D 2B                  DCX   H
341 E11E 34                  INR   M         Token +1 (#A1)
342 E11F 23                  INX   H
343 E120 CDA1E3              CALL  :E3A1     Encode STR expr
344 E123 CD67E8              CALL  :E867     Check if next char is ';'
345 E126 3B                  DATA  :3B
346                                           Into EREAD
347                 *
348                 ****************
349                 * ENCODE 'READ' *
350                 ****************
351                 *
352                 * From L3E16 used by several routines, with another
353                 * address in LXI D, ...., pointing to various kinds
354                 * of encoding/get routines.
355                 *
356 E127 11BCE5      EREAD   LXI   D,:E5BC   Addr routine encode variable
357                                           or array reference
358                 *
359 E12A E5          L3E16   PUSH  H
360 E12B 3600                MVI   M,:00     00 into EBUF (count)
361 E12D CD18E0              CALL  :E018     Update EBUF pointer
362 E130 D5          L3E17   PUSH  D
363 E131 CD64E1              CALL  :E164     Go to encoding routine
364
365                 * Return here after encoding:
366
367 E134 D1                  POP   D
368 E135 E3                  XTHL
369 E136 34                  INR   M         Count in EBUF +1
370 E137 E3                  XTHL
371 E138 CDD2DD              CALL  :DDD2     Get char from line, neglect
372                                           tab + space
373 E13B 0C                  INR   C         Points to next char on line
```

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



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

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

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

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



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

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



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

```
126 E26A 6FE2                DBL   :E26F     Addr encode 'MEM'
127                  *
128 E26C 00                  DATA  :00
129 E26D 72E2                DBL   :E272     Addr encode 'port'
130
131                  * Encode 'MEM':
132
133 E26F 2B          L3E40   DCX   H
134 E270 34                  INR   M         Token +1 (#91)
135 E271 23                  INX   H
136 E272 CD02E3      L3E412  CALL  :E302     Encode <INT expr>,<INT expr>
137 E275 36FF                MVI   M,:FF     FF into EBUF
138 E277 CD18E0              CALL  :E018     Update EBUF pointer
139 E27A CDD2DD              CALL  :DDD2     Get char from line, neglect
140                                          tab + space
141 E27D FE2C                CPI   :2C       ',' ?
142 E27F C0                  RNZ             Ready if not
143 E280 2B                  DCX   H
144 E281 03                  INX   B
145 E282 C36DE3              JMP   :E36D     Encode INT expr
146
147                  * Encode 'TIME':
148
149 E285 2B          L3E41   DCX   H
150 E286 34                  INR   M
151 E287 34                  INR   M         Token +2 (#92)
152 E288 23                  INX   H
153 E289 C36DE3              JMP   :E36D     Encode INT expr
154                  *
155                  **********************************
156                  * ENCODE 'DRAW', 'FILL', 'DOT' *
157                  **********************************
158                  *
159                  EFILL
160 E28C CD02E3      EDRAW   CALL  :E302     Encode <INT expr>,<INT expr>
161
162                  * Entry for encode 'DOT':
163
164 E28F CD02E3      EDOT    CALL  :E302     Idem
165 E292 C314E3              JMP   :E314     Encode INT expr
166                  *
167                  ************************************
168                  * ENCODE 'RUN' AND 'RUN <LINENR>' *
169                  ************************************
170                  *
171 E295 CD59E8      ERUN    CALL  :E859     Next char ':' or 'CR' ?
172 E298 C8                  RZ              Then ready
173
174                  * If 'RUN <linenr>':
175
176 E299 2B                  DCX   H
177 E29A 34                  INR   M         Token +1 (#88)
178 E29B 23                  INX   H
179 E29C C32AE7              JMP   :E72A     Get linenr into EBUF
180                  *
181                  ****************
182                  * ENCODE 'IMP' *
183                  ****************
184                  *
185                  * Note: This command is not encoded, but has
186                  * immediate effect.
187                  *
```



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

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

                        * Entry for encode CLEAR/TALK:

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



```
E31B CD6DE3                     CALL  :E36D     Encode <CHAN>
E31E CD25E3                     CALL  :E325     Encode 'OFF' or <ENV><VOL>
E321 D8                         RC              Ready if 'OFF' given
E322 C311E3                     JMP   :E311     Encode <TG><FREQ>
                        *
                        ****************************************
                        * ENCODE 'NOISE' WITH POSSIBLE 'OFF' *
                        ****************************************
                        *
                        * Encodes NOISE <ENV><VOL> or NOISE OFF.
                        *
                        * Exit: C, HL: Updated.
                        *       CY=1 : Noise off.
                        *       A preserved, B corrupted, D=0, E=1.
                        *
E325 CD2CE3             ENOISE  CALL  :E32C     Encode  possible 'OFF'
E328 D411E3                     CNC   :E311     Encode <ENV><VOL> if no
                                                'OFF' given
E32B C9                         RET
                        *
                        ****************************
                        * ENCODE A POSSIBLE 'OFF' *
                        ****************************
                        *
                        * Exit: CY=1:  'OFF' in input; FF in EBUF.
                        *       CY=0:  No 'OFF' given.
                        *       C, HL: Updated.
                        *       AB preserved, D=0, E=1.
                        *
E32C 1132E3             L3E55   LXI   D,:E332   Startaddr table
E32F C39AE0                     JMP   :E09A     Get addr encoding instr
                                                and go to it

                        * Table:

E332 03                 L3E404  DATA  :03
E333 4F                         DATA  :4F       O
E334 46                         DATA  :46       F
E335 46                         DATA  :46       F
E336 3BE3                       DBL   :E33B     addr encode 'OFF'
                        *
E338 00                         DATA  :00
E339 42E3                       DBL   :E342     encoding addr if no 'OFF'

                        * Encode 'OFF':

E33B 36FF               L3E365  MVI   M,:FF     FF into EBUF
E33D CD18E0                     CALL  :E018     Update EBUF pointer
E340 37                         STC             CY=1
E341 C9                         RET

                        * If no 'OFF':

E342 B7                 L3E366  ORA   A         CY=0
E343 C9                         RET
                        *
                        ******************
                        * ENCODE 'CALLM' *
                        ******************
                        *
E344 CD6DE3             ECALM   CALL  :E36D     Encode memory addr (INT)
E347 36FF                       MVI   M,:FF     FF into EBUF
```

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



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

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



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

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

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

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

                * RGTOP > OLDOP:

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



```
E42A E1                 POP   H
E42B 223901             SHLD  :0139     Restore HOPPT
E42E E1                 POP   H
E42F 223B01             SHLD  :013B     Restore RGTPT
E432 F1                 POP   PSW
E433 323801             STA   :0138     Restore OLDOP
E436 EB                 XCHG            New EBUF pntr in HL
E437 D1                 POP   D         Restore type left operand
                                        (E) and orig RGTOP (D)
E438 7A                 MOV   A,D       Old RGTOP in A
E439 E61F               ANI   :1F       Op.code only
E43B CDCFE7             CALL  :E7CF     Obtain type info for
                                        binary operation
E43E CD57E7             CALL  :E757     Encode binary operation
                                        into EBUF
E441 C304E4             JMP   :E404     Check again if prios correct

                * Unitary operator:

E444 223901     L3E56   SHLD  :0139     Set pntr place for operator
E447 E5                 PUSH  H
E448 CD55E4             CALL  :E455     Encode a term
E44B CD97E7             CALL  :E797     Encode unitary operator
                                        for a term
E44E D1                 POP   D
E44F CD83E7             CALL  :E783     Byte in A into EBUF
E452 C3FEE3             JMP   :E3FE     Encode sequence with prio's
                *
                * ENCODE A TERM:
                *
                * Non-error exit: C,HL: updated.
                *                 BCDE corrupted, AF preserved.
                *
E455 F5         L3E57   PUSH  PSW
E456 CDD2DD             CALL  :DDD2     Get char from line, neglect
                                        tab + space
E459 FE22               CPI   :22
E45B CA9BE4             JZ    :E49B     Jump if char is '"'
E45E FE28               CPI   :28
E460 CA86E4             JZ    :E486     Jump if char is '('
E463 CD02DE             CALL  :DE02     Check if char is upper case
E466 DA77E4             JC    :E477     Then jump
E469 FE2D               CPI   :2D
E46B CA0BDA             JZ    :DA0B     Run 'SYNTAX ERROR' if '-'
E46E CDC8E4             CALL  :E4C8     Encode a number
E471 D20BDA             JNC   :DA0B     Evt run 'SYNTAX ERROR'
E474 C3A4E4             JMP   :E4A4     Store type and quit

                * If upper case character:

E477 CD22E5     L3E58   CALL  :E522     Encode function
E47A DAA4E4             JC    :E4A4     If ready: store type (#30)
                                        and quit
E47D 48                 MOV   C,B
E47E 1600               MVI   D,:00
E480 CDBCE5             CALL  :E5BC     Encode var/array reference
E483 C3A7E4             JMP   :E4A7     Quit

                * If opening bracket:

E486 369A       L3E59   MVI   M,:9A     Load #9A in EBUF
E488 CD18E0             CALL  :E018     Update EBUF pointer
```

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



```
188                  *        HL: Points to EBUF.
189                  *
190                  * On exit: In A:     #00 FPT;  #10 INT;  #10 HEX.
191                  *          In EBUF:  #10 FPT;  #14 INT;  #15 HEX.
192                  *
193                  * On exit, non-hex numbers will only be INT if reqd
194                  * type or IMP type is INT, and there is no '.' or
195                  * 'E' in the input string.
196                  *
197                  * Non-error exit: CY=1: C,HL updated, B preserved.
198                  *                       A: Type-code, DE: entry BC.
199                  * Error exit:     CY=0, BCDEHL preserved.
200                  *
201 E4C8 C5            ENUM  PUSH  B
202 E4C9 E5                  PUSH  H
203 E4CA CDD2DD            CALL  :DDD2      Get char from line, neglect
204                                          tab + space
205 E4CD FE23              CPI   :23
206 E4CF CA13E5            JZ    :E513      Hex if char is '#'
207 E4D2 3A9002            LDA   :0290      Get required number type
208 E4D5 B7                ORA   A
209 E4D6 C2DCE4            JNZ   :E4DC      Jump if reqd type is not FPT
210 E4D9 3A8F02            LDA   :028F      Get default number type
211 E4DC FE00        ENM03 CPI   :00
212 E4DE CAFFE4            JZ    :E4FF      Jump if type is not FPT
213
214                  * If INT:
215
216 E4E1 3614              MVI   M,:14      Load #14 in EBUF
217 E4E3 CD18E0            CALL  :E018      Update EBUF pointer
218 E4E6 CD7BE5            CALL  :E57B      INT nr into EBUF
219 E4E9 D2FFE4            JNC   :E4FF      Then handle as FPT
220 E4EC CDE0DD            CALL  :DDE0      Get char from line
221 E4EF FE2E              CPI   :2E
222 E4F1 CAFFE4            JZ    :E4FF      Try FPT if no digits before
223                                          the '.'
224 E4F4 FE45              CPI   :45
225 E4F6 CAFFE4            JZ    :E4FF      Handle as FPT if 'E'
226 E4F9 3E10        ENM05 MVI   A,:10      Type is INT
227 E4FB D1          ENM10 POP   D
228 E4FC D1                POP   D
229 E4FD 37                STC               CY=1
230 E4FE C9                RET
231
232                  * If FPT:
233
234 E4FF E1          ENM20 POP   H
235 E500 C1                POP   B
236
237                  * Entry from ETCON:
238
239 E501 C5          ENM25 PUSH  B            ) Save pointers
240 E502 E5                PUSH  H            )
241 E503 3610              MVI   M,:10      Load #10 in EBUF
242 E505 CD18E0            CALL  :E018      Update EBUF pointer
243 E508 CD96E5            CALL  :E596      FPT nr into EBUF
244 E50B 3E00              MVI   A,:00      Type is FPT
245 E50D DAFBE4            JC    :E4FB      Jump if no errors
246 E510 E1                POP   H
247 E511 C1                POP   B
248 E512 C9                RET              Error exit (no number)
249
```

```
                        * If HEX:
                        *
E513 3615       ENM30   MVI   M,:15      Load #15 in EBUF
E515 CD18E0             CALL  :E018      Update EBUF pointer
E518 00                 NOP
E519 CD84E8             CALL  :E884      Hex nr into EBUF
E51C D20BDA             JNC   :DA0B      Run 'SYNTAX ERROR' if no
                                         digits
E51F C3F9E4             JMP   :E4F9      Quit
                        *
                        **********************
                        * ENCODE A FUNCTION *
                        **********************
                        *
                        * Reads a system function (both function and
                        * arguments) into EBUF. Error exit if syntax or
                        * type mismatch errors are found.
                        *
                        * Entry:   HL: Points to 1st free pos. in EBUF.
                        *          C : Points to input.
                        * Exit: If found: CY=1:
                        *          A:  Type info of result.
                        *          C,HL updated; BDE corrupted.
                        *       If not found: CY=0:
                        *          A:  End of table count.
                        *          B:  Start of name in input.
                        *          C:  Points beyond.
                        *          DE: Points to 0 T/L byte at table end.
                        *          HL: Points to EBUF.
                        *
E522 E5         EFUN    PUSH  H
E523 37                 STC              Set 'include type letter'
E524 CDFDE6             CALL  :E6FD      Find variable name in input,
                                         allow !,%,$.
E527 21E6CF             LXI   H,:CFE6    Addr table BASIC functions
E52A CD5ACA             CALL  :CA5A      Find function in table
E52D 7B                 MOV   A,E        Get serialnr of entry in
                                         table in A
E52E EB                 XCHG
E52F E1                 POP   H
E530 D27AE5             JNC   :E57A      Abort if not found (CY=0)

                        * If found in table:

E533 3620               MVI   M,:20      Load fn.code (#20) in EBUF
E535 CD18E0             CALL  :E018      Update EBUF pointer
E538 77                 MOV   M,A        Load how manyth function
                                         in EBUF
E539 CD18E0             CALL  :E018      Update EBUF pointer
E53C 1A                 LDAX  D          Get T/L byte of function
E53D 13                 INX   D
E53E F5                 PUSH  PSW        Preserve it
E53F E60F               ANI   :0F        Le ngth only
E541 CA76E5             JZ    :E576      Jump if no arguments reqd

                        * If arguments required:

E544 F5                 PUSH  PSW        Preserve length fuction
E545 CD67E8             CALL  :E867      Check if next char is '(',
E548 28                 DATA  :28        run 'SYNTAX ERROR' if not
E549 EB         L3E72   XCHG
E54A 7E                 MOV   A,M        Get T/L byte
```



```
E54B 23                 INX   H
E54C EB                 XCHG
E54D FE30               CPI   :30        Boolean type ?
E54F CCBCE5             CZ    :E5BC      Then encode var/array ref.
E552 CA65E5             JZ    :E565      and jump
E555 FE20               CPI   :20        STR type ?
E557 CCA1E3             CZ    :E3A1      Then encode STR expr
E55A CA65E5             JZ    :E565      and jump
E55D FE10               CPI   :10        INT type ?
E55F CC6DE3             CZ    :E36D      Then encode INT expr
E562 C476E3             CNZ   :E376      Else: encode FPT expr
                        *
E565 F1         L3E73   POP   PSW        Get length function
E566 3D                 DCR   A          Check if all arguments done
E567 CA72E5             JZ    :E572      Jump if ready
E56A F5                 PUSH  PSW        Preserve T/L function
E56B CD67E8             CALL  :E867      Check if next char is ','
E56E 2C                 DATA  :2C        run 'SYNTAX ERROR' if not
E56F C349E5             JMP   :E549      Encode next argument
E572 CD67E8     L3E74   CALL  :E867      Check if next char is ')',
E575 29                 DATA  :29        run 'SYNTAX ERROR' if not
E576 F1         L3E75   POP   PSW        Get T/L function
E577 E630               ANI   :30        Type only
E579 37                 STC              CY=1
E57A C9         L3E76   RET
                        *
                        *************************
                        * INT NUMBER INTO EBUF *
                        *************************
                        *
                        * Entry: C : Points to input.
                        *        HL: Points to EBUF.
                        *
E57B CDD2DD     EINT    CALL  :DDD2      Get char from line, neglect
                                         tab + space
E57E FE2D               CPI   :2D
E580 C285E5             JNZ   :E585      Jump if char is not '-'
E583 0C                 INR   C
E584 AF                 XRA   A          Else: clear sign bit
E585 CD24C0     L3E78   CALL  :C024       Input INT number to MACC
E588 C28DE5             JNZ   :E58D      Jump if nr was >= 0
E58B E7                 RST   4          Change sign MACC
E58C 60                 DATA  :60
E58D C3A3E5     L3E79   JMP   :E5A3       Move MACC into EBUF
                        *
                        *************************
                        * HEX NUMBER INTO EBUF *
                        *************************
                        *
E590 CD2AC0     L3E80   CALL  :C02A       Input HEX number to MACC
E593 C3A3E5             JMP   :E5A3      Move MACC into EBUF
                        *
                        ************************
                        * FPT NUMBER INTO EBUF *
                        ************************
                        *
                        * Entry: C : Points to input.
                        *        HL: Points to EBUF.
                        * Non-error exit:  CY=1:
                        *        C,HL updated. B preserved, A corrupted.
                        *        DE: Location of number in EBUF.
                        * Error exit: CY=0:
```

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

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

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



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

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



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

```
                        * The variable name is inserted in the symbol table
                        * and the value is cleared.
                        *
                        * Entry: See CAB8.
                        * Exit:  HL: Points to 2nd T/L byte of entry.
                        *        AF corrupted, BCDE preserved.
                        *
E67D CDB8CA     EVARI   CALL  :CAB8      Insert var.name in symtab
E680 E5                 PUSH  H
E681 23                 INX   H         HL pnts after 2nd T/L byte
                                        of entry
E682 7A                 MOV   A,D       Get T/L of name
E683 E640               ANI   :40
E685 C295E6             JNZ   :E695     Jump if array type

                        * If number type:

E688 7A                 MOV   A,D       Get T/L of name
E689 E630               ANI   :30
E68B FE20               CPI   :20
E68D CA95E6             JZ    :E695     Jump if string type
E690 CD9ECB             CALL  :CB9E     Clear value in symtab
E693 E1                 POP   H
E694 C9                 RET

                        * If string/array type:

E695 3600       EVI10   MVI   M,:00     ) Clear pointer in symbtab
E697 23                 INX   H         )
E698 3600               MVI   M,:00     )
E69A E1                 POP   H
E69B C9                 RET
                        *
                        ******************************
                        * STORE QUOTED TEXT IN EBUF *
                        ******************************
                        *
E69C 3618       L3E96   MVI   M,:18     Code for quoted string (#18)
                                        into EBUF
E69E CD18E0             CALL  :E018     Update EBUF pointer
E6A1 1EFF               MVI   E,:FF     Text must end with '"'
E6A3 C3B5E6             JMP   :E6B5     Into common end
                        *
                        **********************************
                        * STORE UNQUOTED STRING IN EBUF *
                        **********************************
                        *
E6A6 1E01       L3E97   MVI   E,:01     Text must end with ','
E6A8 3619               MVI   M,:19     Code for unquoted string
                                        (#19) into EBUF
E6AA CD18E0             CALL  :E018     Update EBUF pointer
E6AD C3B5E6             JMP   :E6B5     Into common end
                        *
                        ************************
                        * STORE TEXT INTO EBUF *
                        ************************
                        *
                        * Text in DATA, REM and '***' statements is moved
                        * into the EBUF.
                        *
E6B0 CDD2DD     L3E98   CALL  :DDD2     Get char from line, neglect
                                        tab + space
```



```
E6B3 1E02               MVI   E,:02     Text must end with CR
                                        Into common end
                        *
                        ************************************
                        * COMMON END TEXT ENCODING ROUTINES *
                        ************************************
                        *
                        * Entry: C : Points to 1st actual character to be
                        *            stored.
                        *        HL: Points to place for length byte in EBUF
                        *        E : Handling switch:
                        *             > 1: (but <#80): Text must end with CR
                        *             = 1: Text will end with ',' (',' is no
                        *                  inserted into EBUF).
                        *            <= 0: Text will end at '"' ('"' is not
                        *                  inserted into the EBUF).
                        * Exit:  C : Points beyond text in input.
                        *        HL: Points beyond stored text in EBUF.
                        *        D : Length of stored text.
                        *        A : Character which marks end of text.
                        *        B preserved, E corrupted.
                        *
E6B5 E5         L3E99   PUSH  H
E6B6 CD18E0             CALL  :E018     Update EBUF pointer
E6B9 1600               MVI   D,:00     Set length is 0
E6BB CDE0DD     L3E100  CALL  :DDE0     Get char from line
E6BE FE0D               CPI   :0D
E6C0 CADBE6             JZ    :E6DB     Jump if char is 'CR'
E6C3 FE2C               CPI   :2C
E6C5 CAE3E6             JZ    :E6E3     Jump if char is ','
E6C8 0C         L3E101  INR   C
E6C9 FE22               CPI   :22
E6CB C2D3E6             JNZ   :E6D3     Jump if char is not '"'
E6CE 1D                 DCR   E
E6CF FADFE6             JM    :E6DF     If done: store length in
                                        EBUF, quit
E6D2 1C                 INR   E
                        *
                        * Character into EBUF:
                        *
E6D3 77         L3E102  MOV   M,A       Load char in EBUF
E6D4 CD18E0             CALL  :E018     Update EBUF pointer
E6D7 14                 INR   D
E6D8 C3BBE6             JMP   :E6BB     Get next char
                        *
                        * If 'CR':
                        *
E6DB 1D         L3E103  DCR   E
E6DC FA0BDA             JM    :DA0B     If E >= #80: Run 'SYNTAX
                                        ERROR'
E6DF E3         L3E104  XTHL
E6E0 72                 MOV   M,D       Length in EBUF entry
E6E1 E1                 POP   H
E6E2 C9                 RET
                        *
                        * If ',':
                        *
E6E3 1D         L3E105  DCR   E
E6E4 CADFE6             JZ    :E6DF     If E=0: Store length in
                                        EBUF, quit
E6E7 C3ABE8             JMP   :E8AB     incr E, get next char
                        *
```

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

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

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



```
                        ORG   :E6FD
                        *
                        *
                        *
                        ********************************
                        * FIND VARIABLE NAME IN INPUT *
                        ********************************
                        *
                        * Checks if 1st character is a upper case one.
                        * Reads the input (starting with character after
                        * C) till it finds a non-alphanumeric character
                        * (number or upper case). On a carry CALL, it also
                        * accepts %, ! or $ at the end. Blanks are not
                        * ignored and not accepted.
                        *
                        * Entry: C :  Input position.
                        * Exit:  B :  Entry C.
                        *        C :  Points to 1st character not accepted.
                        *        D :  Count of 1st character not accepted.
                        *             (1st character read has count 1).
                        *        A,E: 1st non-alphanumeric character read.
                        *        HL preserved, F corrupted.
                        *
E6FD F5         RDID    PUSH  PSW       Preserve CY-flag
E6FE 41                 MOV   B,C
E6FF 1600               MVI   D,:00     Init count
E701 14         L3E110  INR   D         Count +1
E702 0C                 INR   C         Line pos +1
E703 7A                 MOV   A,D       Count in A
E704 FE0F               CPI   :0F       Max 14 char for a name
E706 DA0AE7             JC    :E70A
E709 15                 DCR   D         Skip last char if > 14
E70A CDE0DD     L3E111  CALL  :DDE0     Get char from line
E70D CD09DE             CALL  :DE09     Check if nr or upper case
E710 DA01E7             JC    :E701     Get next char if O.K.
E713 CDE0DD             CALL  :DDE0     Get 1st non-alphanum.char
                                        from line
E716 5F                 MOV   E,A       Store it in E
E717 F1                 POP   PSW       Get CY-flag back
E718 7B                 MOV   A,E       Get char back in A
E719 D0                 RNC             Abort if non-carry CALL

                        * On carry CALL only: accept !,%,$:

E71A FE25               CPI   :25
E71C CA27E7             JZ    :E727     Jump if char is '%'
E71F FE21               CPI   :21
E721 CA27E7             JZ    :E727     Jump if char is '!'
E724 FE24               CPI   :24
E726 C0                 RNZ             Abort if char is not '$'
E727 0C         L3E112  INR   C         Update line pos
E728 14                 INR   D         Update count
E729 C9                 RET
                        *
                        ********************
                        * GET LINE NUMBER *
                        ********************
                        *
                        * Exit: C,HL: Updated.
                        *       B preserved, AFDE corrupted.
                        *       'SYNTAX ERROR' if no linenr given.
```

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



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

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

                        * If 'INOT':

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

                        * If '+' or '-':

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



```
                        * type.
                        * Type of result is stored in TYPE.
                        *
                        * Exit: E : Type code from table.
                        *       A : Code for EBUF:  1xx xxxxx:
                        *           bits 5,6: type of compute required:
                        *                     0 : FPT
                        *                     1 : INT
                        *                     2 : STR
                        *                     3 : Boolean
                        *           bits 0-4: Opcode.
                        *       BCDHL preserved.
                        *
E7CF E5         L3E125  PUSH  H
E7D0 D5                 PUSH  D
E7D1 E61F               ANI   :1F       Opcode only
E7D3 F5                 PUSH  PSW
E7D4 CD1FE8             CALL  :E81F     Set D according to opcode
E7D7 3A3601             LDA   :0136     Get type latest expression
E7DA BB                 CMP   E         Compare both types
E7DB C209E8             JNZ   :E809     Jump if not identical
E7DE 07                 RLC             )
E7DF 07                 RLC             ) Type code from TYPE in
E7E0 07                 RLC             ) lonibble
E7E1 07                 RLC             )
E7E2 E603               ANI   :03       Only lower 2 bits
E7E4 6F                 MOV   L,A       in L
E7E5 7A         L3E126  MOV   A,D       Get opcode group (0-5)
E7E6 87                 ADD   A         *2
E7E7 57                 MOV   D,A       in D
E7E8 87                 ADD   A         *4
E7E9 82                 ADD   D         *6 (find group in table)
E7EA 85                 ADD   L         Find pos in grouptable
E7EB 2135E8             LXI   H,:E835   Startaddr result table
E7EE CD30DE             CALL  :DE30     Find addr resultcode in
                                        table
E7F1 7E                 MOV   A,M       Get resultcode
E7F2 3C                 INR   A         Check if code is FF
E7F3 CA1ADA             JZ    :DA1A     Then run error 'TYPE
                                        MISMATCH'
E7F6 3D                 DCR   A
E7F7 E630               ANI   :30       Get type of result only
E7F9 323601             STA   :0136     Store type latest expression
E7FC D1                 POP   D         Get code for binary
                                        operation in D
E7FD 5E                 MOV   E,M       ) Get resultcode from table
E7FE 7E                 MOV   A,M       ) in E and in A
E7FF 1F                 RAR
E800 E660               ANI   :60       Reqd computing in bits 5,6
E802 B2                 ORA   D         Add opcode in bits 0-4
E803 F680               ORI   :80       Set bit 7
E805 E1                 POP   H
E806 54                 MOV   D,H       Restore D
E807 E1                 POP   H
E808 C9                 RET

                        * If both types not identical:

E809 2E04       L3E127  MVI   L,:04
E80B FE00               CPI   :00       TYPE is FPT ?
E80D CA16E8             JZ    :E816     Then jump
E810 2C                 INR   L
```

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



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

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

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

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



```
064                                             without arguments
065 E876 C32AE1                 JMP   :E12A     Encode
066                     *
067                     ******************************
068                     * INPUT FPT NUMBER INTO MACC *
069                     ******************************
070                     *
071                     * Entry: Z=1: Change sign too.
072                     * Exit:  C updated, ABDEHL preserved.
073                     *        CY=0: Error.
074                     *
075 E879 CD1EC0         L3E134  CALL  :C01E     Input FPT number to MACC
076 E87C C0                    RNZ             Ready if Z=0
077 E87D E7                    RST   4         Else change sign MACC
078 E87E 1B                    DATA  :1B
079 E87F C9                    RET
080                     *
081                     **********************************
082                     * STORE QUOTED TEXT INTO EBUF *
083                     **********************************
084                     *
085                     * Entry: C points to 1st '"'.
086                     *
087 E880 0C             L3E135  INR   C
088 E881 C39CE6                 JMP   :E69C     Store text in EBUF
089                     *
090                     **********************************
091                     * STORE A HEX NUMBER INTO EBUF *
092                     **********************************
093                     *
094                     * Entry: C points to '#' of hex number.
095                     *
096 E884 0C             EHEX    INR   C
097 E885 C390E5                 JMP   :E590     Hex nr into EBUF
098                     *
099                     ************************************
100                     * ENCODE AN INT NUMBER INTO EBUF *
101                     ************************************
102                     *
103 E888 CDAFE8         L3E137  CALL  :E8AF     #14 into EBUF
104 E88B FE23                  CPI   :23       '#' ?
105 E88D CA99E8                 JZ    :E899     Then jump
106 E890 CD7BE5                 CALL  :E57B     INT nr into EBUF
107 E893 D2B7E8         L3E139  JNC   :E8B7     Evt run 'SYNTAX ERROR'
108 E896 C35CE1                 JMP   :E15C     Quit
109
110                     * If hex number:
111
112 E899 CD84E8         L3E138  CALL  :E884     Hex nr into EBUF
113 E89C C393E8                 JMP   :E893
114                     *
115 E89F FF                     DATA  :FF
116 E8A0 FF                     DATA  :FF
117 E8A1 FF                     DATA  :FF
118                     *
119                     ******************
120                     * ENCODE 'DATA' *
121                     ******************
122                     *
123 E8A2 7D             EDATA   MOV   A,L
124 E8A3 FE42                   CPI   :42
125 E8A5 C20BDA                 JNZ   :DA0B     Run 'SYNTAX ERROR' if not
```

```
E8A8 C366E3            JMP   :E366     Encode text
                *
                ********************************
                * part of END ENCODING (3E6B5) *
                ********************************
                *
E8AB 1C        L3E140  INR   E
E8AC C3C8E6            JMP   :E6C8
                *
                ********************************
                * CODE FOR INT NUMBER INTO EBUF *
                ********************************
                *
                * Gets also next character to encode.
                *
E8AF 3614      L3E141  MVI   M,:14     INT code (#14) in EBUF
E8B1 CD18E0            CALL  :E018     Update EBUF pointer
E8B4 C3D2DD            JMP   :DDD2     Get char from line, neglect
                                       tab + space
                *
                ************************************************
                * ERROR EXIT OF ENCODE INT NR INTO EBUF (3E888) *
                ************************************************
                *
E8B7 2A3201    L3E142  LHLD  :0132     Get EFEPT
E8BA 11FCFF            LXI   D,:FFFC
E8BD 19                DAD   D          Set back linepntr
E8BE 220001            SHLD  :0100     Store start current line
E8C1 C30BDA            JMP   :DA0B     Run 'SYNTAX ERROR'
                *
E8C4 FF                DATA  :FF
                *
                ***************************************
                * ASCII TABLE UPPER CASE (UNSHIFTED) *
                ***************************************
                *
E8C5 30        KEYTU   DATA  :30       0
E8C6 31                DATA  :31       1
E8C7 32                DATA  :32       2
E8C8 33                DATA  :33       3
E8C9 34                DATA  :34       4
E8CA 35                DATA  :35       5
E8CB 36                DATA  :36       6
E8CC 37                DATA  :37       7
E8CD 38                DATA  :38       8
E8CE 39                DATA  :39       9
E8CF 3A                DATA  :3A       :
E8D0 3B                DATA  :3B       ;
E8D1 2C                DATA  :2C       ,
E8D2 2D                DATA  :2D       -
E8D3 2E                DATA  :2E       .
E8D4 2F                DATA  :2F       /
E8D5 0D                DATA  :0D       car ret
E8D6 41                DATA  :41       A
E8D7 42                DATA  :42       B
E8D8 43                DATA  :43       C
E8D9 44                DATA  :44       D
E8DA 45                DATA  :45       E
E8DB 46                DATA  :46       F
E8DC 47                DATA  :47       G
E8DD 48                DATA  :48       H
E8DE 49                DATA  :49       I
```



```
E8DF 4A                DATA  :4A       J
E8E0 4B                DATA  :4B       K
E8E1 4C                DATA  :4C       L
E8E2 4D                DATA  :4D       M
E8E3 4E                DATA  :4E       N
E8E4 4F                DATA  :4F       O
E8E5 50                DATA  :50       P
E8E6 51                DATA  :51       Q
E8E7 52                DATA  :52       R
E8E8 53                DATA  :53       S
E8E9 54                DATA  :54       T
E8EA 55                DATA  :55       U
E8EB 56                DATA  :56       V
E8EC 57                DATA  :57       W
E8ED 58                DATA  :58       X
E8EE 59                DATA  :59       Y
E8EF 5A                DATA  :5A       Z
E8F0 5B                DATA  :5B       [
E8F1 5E                DATA  :5E       ^
E8F2 20                DATA  :20       space
E8F3 00                DATA  :00       (rept)
E8F4 08                DATA  :08       char del
E8F5 10                DATA  :10       cursor up
E8F6 11                DATA  :11       cursor down
E8F7 12                DATA  :12       cursor left
E8F8 13                DATA  :13       cursor right
E8F9 09                DATA  :09       tab
E8FA 80                DATA  :80       ctrl
E8FB 00                DATA  :00       (break)
E8FC 00                DATA  :00       (shift)
                *
                ***************************************
                * ASCII TABLE LOWER CASE (SHIFTED) *
                ***************************************
                *
E8FD 30        KEYTS   DATA  :30       0
E8FE 21                DATA  :21       !
E8FF 22                DATA  :22       "
E900 23                DATA  :23       #
E901 24                DATA  :24       $
E902 25                DATA  :25       %
E903 26                DATA  :26       &
E904 27                DATA  :27       '
E905 28                DATA  :28       (
E906 29                DATA  :29       )
E907 2A                DATA  :2A       *
E908 2B                DATA  :2B       +
E909 3C                DATA  :3C       <
E90A 3D                DATA  :3D       =
E90B 3E                DATA  :3E       >
E90C 3F                DATA  :3F       ?
E90D 0D                DATA  :0D       car ret
E90E 61                DATA  :61       a
E90F 62                DATA  :62       b
E910 63                DATA  :63       c
E911 64                DATA  :64       d
E912 65                DATA  :65       e
E913 66                DATA  :66       f
E914 67                DATA  :67       g
E915 68                DATA  :68       h
E916 69                DATA  :69       i
E917 6A                DATA  :6A       j
```

```
E918 6B                 DATA  :6B       k
E919 6C                 DATA  :6C       l
E91A 6D                 DATA  :6D       m
E91B 6E                 DATA  :6E       n
E91C 6F                 DATA  :6F       o
E91D 70                 DATA  :70       p
E91E 71                 DATA  :71       q
E91F 72                 DATA  :72       r
E920 73                 DATA  :73       s
E921 74                 DATA  :74       t
E922 75                 DATA  :75       u
E923 76                 DATA  :76       v
E924 77                 DATA  :77       w
E925 78                 DATA  :78       x
E926 79                 DATA  :79       y
E927 7A                 DATA  :7A       z
E928 5D                 DATA  :5D       ]
E929 7E                 DATA  :7E       ~
E92A 20                 DATA  :20       space
E92B 00                 DATA  :00       (rept)
E92C 08                 DATA  :08       char del
E92D 14                 DATA  :14       window up
E92E 15                 DATA  :15       window down
E92F 16                 DATA  :16       window left
E930 17                 DATA  :17       window right
E931 09                 DATA  :09       tab
E932 80                 DATA  :80       ctrl
E933 00                 DATA  :00       (break)
E934 00                 DATA  :00       (shift)
                *
                ************************************
                * GET INPUTS FROM KEYBOARD OR DINC *
                ************************************
                *
                * Part of RESET (C719). Determines input source
                * depending on 1st input done.
                *
E935 CDBBD6     L3E143  CALL  :D6BB     Scan keyb; char in A
E938 D8                 RC              Ready if break pressed
E939 C0                 RNZ             Ready if key input done
E93A C3F4EF             JMP   :EFF4     Else: Get input from DINC
                *
E93D FF                 DATA  :FF
E93E FF                 DATA  :FF
                *
                ************************************************
                * LOAD ASCII VALUE FOR KEY PRESSED IN BUFFER *
                ************************************************
                *
                * From the key pressed, the offset to the start-
                * address of the ASCII table is calculated. The
                * ASCII value for the pressed key is stored in
                * the circular buffer KLIND.
                *
                * Entry: B : Column number.
                *        C : Row number.
                * Exit:  All registers preserved.
                *
E93F F5         INKEY   PUSH  PSW
E940 C5                 PUSH  B
E941 E5                 PUSH  H
E942 3E07               MVI   A,:07     )
```



```
E944 90                 SUB   B         )
E945 87                 ADD   A         ) Calc offset of startaddr
E946 87                 ADD   A         ) for key pressed.
E947 87                 ADD   A         )   Store it in C
E948 81                 ADD   C         )
E949 4F                 MOV   C,A       )
E94A 2AA702             LHLD  :02A7     Get startaddr ASCII table
E94D 0600               MVI   B,:00
E94F FE11               CPI   :11       )
E951 DA5DE9             JC    :E95D     ) Check if key is a char
E954 FE2B               CPI   :2B       ) A-Z
E956 D25DE9             JNC   :E95D     )
E959 3AC302             LDA   :02C3     Get shift lock value
E95C 47                 MOV   B,A       in B
E95D 3AB002     L3E145  LDA   :02B0     Get 'shift' byte
E960 A8                 XRA   B         Take CTRL into account
E961 E640               ANI   :40       A=#40 if shift, 00 when not
E963 CA6BE9             JZ    :E96B     Jump if no shift
E966 D5                 PUSH  D
E967 113800             LXI   D,:0038   Add. offset for lower case
                                        table
E96A 19                 DAD   D         Startaddr lower case table
                                        now in HL
E96B D1                 POP   D
E96C 0600       L3E146  MVI   B,:00
E96E 09                 DAD   B         Add offset to startaddr
E96F 7E                 MOV   A,M       Get ASCII value from table
E970 B7                 ORA   A         Check if Break, Rept, Shift
E971 CA8EE9             JZ    :E98E     Then Pop, ret
E974 FE80               CPI   :80       Check if CTRL
E976 CA92E9             JZ    :E992     Then update CTRL flag
E979 47                 MOV   B,A       Store ASCII value in B
E97A 2ABE02             LHLD  :02BE     Get addr next pos in KLIND
E97D E5                 PUSH  H         Store KLIIN on stack
E97E CD9CD6             CALL  :D69C     Update KLIND pointer
E981 3AC002             LDA   :02C0     Get lsbyte next output pos
                                        of KLIND
E984 BD                 CMP   L         Compare with KLIIN
E985 CA8DE9             JZ    :E98D     Abort if buffer full
E988 22BE02             SHLD  :02BE     Update KLIIN
E98B E3                 XTHL            Get old KLIIN from stack
E98C 70                 MOV   M,B       Store ASCII char in KLIND
E98D E1         L3E147  POP   H
E98E E1         L3E148  POP   H
E98F C1                 POP   B
E990 F1                 POP   PSW
E991 C9                 RET
                *
                * Update CTRL flag:
                *
E992 3AC302     L3E149  LDA   :02C3     Get shiftlock value
E995 2F                 CMA             Invert it
E996 32C302             STA   :02C3     And store it again
E999 C38EE9             JMP   :E98E     Pop, ret
                *
                ****************
                * HEAP REQUEST *
                ****************
                *
                * The routine checks the heap for free areas. Evt.
                * consecutive free areas are consolidated. If this
                * procedure finds a free area min. 2 bytes larger
```

```
                        * than requested, then it is reserved by setting
                        * the length bytes (msb=0). An evt. resting free
                        * area is set with length bytes and msb=1 (this
                        * area must be >= 2 bytes).
                        * The heap contents is never moved to obtain one
                        * large consolidated area of free bytes!!
                        *
                        * Entry: DE: Length requested heap space.
                        * Exit:  AFBCDE preserved.
                        *        HL: Points to a 2-byte length of the re-
                        *            quested gap. If no space available,
                        *            it points to an error routine.
                        *
E99C F5                 HREQ    PUSH  PSW
E99D C5                         PUSH  B
E99E D5                         PUSH  D
E99F 42                         MOV   B,D       ) Reqd length in BC
E9A0 4B                         MOV   C,E       )
E9A1 2A9B02                     LHLD  :029B     Get startaddr Heap
E9A4 56                 HR010   MOV   D,M       )
E9A5 23                         INX   H         ) Contents 1st 2 bytes of
E9A6 5E                         MOV   E,M       ) Heap in DE (length)
E9A7 23                         INX   H         )
E9A8 7A                         MOV   A,D       1st byte in A
E9A9 E67F                       ANI   :7F       Mask bit 'free/used'
E9AB BA                         CMP   D
E9AC 57                         MOV   D,A       D is length without msb
E9AD CA27D2                     JZ    :D227     If area not free: check if
                                                end of heap reached. JMP
                                                #E9FA if not

                        * If free area found: Check all next heap entries
                        * and accumulate all free areas in succession:

E9B0 E5                 HR020   PUSH  H         Save startaddr area +2
E9B1 19                         DAD   D         Begin next area in HL
E9B2 7E                         MOV   A,M
E9B3 B7                         ORA   A         Check msb of this area
E9B4 F2C7E9                     JP    :E9C7     Jump if area occupied
E9B7 23                         INX   H
E9B8 7B                         MOV   A,E
E9B9 86                         ADD   M         Add lobyte length next area
                                                length previous area
E9BA 5F                         MOV   E,A
E9BB 7A                         MOV   A,D
E9BC 2B                         DCX   H
E9BD 8E                         ADC   M         Add hibyte length next area
                                                length previous area
E9BE E67F                       ANI   :7F       Skip bit 'free/occupied'
E9C0 57                         MOV   D,A
E9C1 13                         INX   D
E9C2 13                         INX   D         Add 2 exta bytes
E9C3 E1                         POP   H         Restore start free area +2
E9C4 C3B0E9                     JMP   :E9B0     Check next area

                        * Next area is not free:

E9C7 E1                 HR030   POP   H         Restore start free area +2
E9C8 E5                         PUSH  H
E9C9 2B                         DCX   H
E9CA 73                         MOV   M,E       )
E9CB 2B                         DCX   H         ) Store total free length in
```



```
E9CC 7A                         MOV   A,D       ) 1st 2 bytes of free area
E9CD F680                       ORI   :80       )
E9CF 77                         MOV   M,A       )
                                                Space available here: HL pnt
                                                to start free area +2; DE is
                                                size free area; BC size reqd
E9D0 7B                         MOV   A,E       )
E9D1 91                         SUB   C         )
E9D2 6F                         MOV   L,A       ) Calc free length - reqd.
E9D3 7A                         MOV   A,D       ) length; result in HL
E9D4 98                         SBB   B         )
E9D5 67                         MOV   H,A       )
E9D6 FAF9E9                     JM    :E9F9     Space not sufficient: Leave
                                                consolidated area as free
E9D9 C2E4E9                     JNZ   :E9E4     Jump if sufficient space
E9DC B5                         ORA   L
E9DD CAF0E9                     JZ    :E9F0     Jump if just enough
E9E0 3D                         DCR   A
E9E1 CAF9E9                     JZ    :E9F9     Not useable if 1 free byte
                                                left

                        * Set not used part of free area to free:

E9E4 EB                 HR040   XCHG            Addr free area in DE
E9E5 1B                         DCX   D
E9E6 1B                         DCX   D         Reserve 2 bytes for length
E9E7 E1                         POP   H         Restore start free area +2
E9E8 E5                         PUSH  H
E9E9 09                         DAD   B         Add reqd length to find
                                                start of resting free area
E9EA 7A                         MOV   A,D
E9EB F680                       ORI   :80       Set free flag
E9ED 77                         MOV   M,A       )
E9EE 23                         INX   H         ) Length free area into heap
E9EF 73                         MOV   M,E       )

                        * Reserve area for requested entry:

E9F0 E1                 HR050   POP   H         Restore start free area +2
E9F1 2B                         DCX   H
E9F2 71                         MOV   M,C       )
E9F3 2B                         DCX   H         ) Reqd length in 1st 2 bytes
E9F4 70                         MOV   M,B       )
E9F5 D1                         POP   D
E9F6 C1                         POP   B
E9F7 F1                         POP   PSW
E9F8 C9                         RET

                        * Area too small:

E9F9 E1                 HR070   POP   H         Restore start free area
E9FA 19                 HR075   DAD   D         HL pnts to next area
E9FB C3A4E9                     JMP   :E9A4     Check next area
                        *
E9FE FF                         DATA  :FF
E9FF FF                         DATA  :FF
                        *
                        *
                        *
                        *
EA00                            END
```

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

ECHRI  E867   EDATA  E8A2   EHEX   E884   HRE0   E99C
HR010  E9A4   HR020  E9B0   HR030  E9C7   HR040  E9E4
HR050  E9F0   HR070  E9F9   HR075  E9FA   INKEY  E93F
KEYTS  E8FD   KEYTU  E8C5   L3E131 E862   L3E133 E873
L3E134 E879   L3E135 E880   L3E137 E888   L3E138 E899
L3E139 E893   L3E140 E8AB   L3E141 E8AF   L3E142 E8B7
L3E143 E935   L3E145 E95D   L3E146 E96C   L3E147 E98D
L3E148 E98E   L3E149 E992   TSEOC  E859
```



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

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



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

```
                        * to be displayed. Break will abort the print out.
                        *
                        * Exit: CY=1, All registers corrupted.
                        *
EAB3 0E02               DISPK   MVI   C,:02      Nr of addr inputs required
EAB5 CDDEEA                     CALL  :EADE      Get laddr and haddr on stack
EAB8 0D                         DCR   C
EAB9 F262EA                     JP    :EA62      Error if only 1 addr given
EABC D1                         POP   D          Haddr in DE
EABD E1                         POP   H          Laddr in HL

                        * Direct call entry:

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



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

                        * Check for delimiter/terminator:

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

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

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

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

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

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



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

```
126 EBA5 7E                MOV   A,M        Get instr code
127 EBA6 A0                ANA   B          ) Check if it is a
128 EBA7 FEC4              CPI   :C4        ) conditional CALL
129 EBA9 C2B0EB            JNZ   :EBB0      Jump if not
130 EBAC 23          L3E183 INX   H
131 EBAD 23                INX   H
132 EBAE 23          L3E184 INX   H
133 EBAF E3                XTHL             Addr next instr on stack
134                                         if it was a RST or CALL
135
136                  * Check if current instruction is inside window:
137
138 EBB0 210000      L3E185 LXI   H,:0000
139 EBB3 39                DAD   SP
140 EBB4 225B00            SHLD  :005B      Save SP
141 EBB7 CDE7ED            CALL  :EDE7      Exchange bytes in reg. save
142                                         area
143 EBBA 2A5100            LHLD  :0051      Get addr current instr
144 EBBD EB                XCHG             in DE
145 EBBE 2A4A00            LHLD  :004A      Get laddr window
146 EBC1 CD45EC            CALL  :EC45      Compare DE-HL
147 EBC4 FADCEB            JM    :EBDC      Jump if addr outside window
148 EBC7 2A4800            LHLD  :0048      Get haddr window
149 EBCA CD45EC            CALL  :EC45      Compare DE-HL
150 EBCD D2DCEB            JNC   :EBDC      Jump if addr outside window
151
152                  * Print registers contents if address inside window:
153
154 EBD0 CD3AED              CALL  :ED3A    Print car.ret
155 EBD3 115100            LXI   D,:0051    Startaddr reg. save area
156 EBD6 219CEE            LXI   H,:EE9C    Startaddr symbol table
157 EBD9 CD44EE            CALL  :EE44      Print reg. contents
158 EBDC CDC2EE      L3E186 CALL  :EEC2      Scan for Break pressed;
159                                         evt run Break
160 EBDF 2A5D00            LHLD  :005D      Get addr next instr
161
162                  * Disable UT to trace itself:
163                  * Entry from init, RST0.
164                  * HL is address where to continu.
165
166 EBE2 3EEA        L3E187 MVI   A,:EA
167 EBE4 BC                CMP   H          Check if hibyte = EA
168 EBE5 C2EEEB            JNZ   :EBEE      Jump if not
169 EBE8 3E0D              MVI   A,:0D
170 EBEA BD                CMP   L          Check if lobyte = 0D
171 EBEB CA62EA            JZ    :EA62      Then go to Error
172
173                  * Check if next opcode is RST or EI/DI:
174
175 EBEE F3          L3E188 DI
176 EBEF 225100            SHLD  :0051      Save addr current instr
177 EBF2 114C00            LXI   D,:004C
178 EBF5 EB                XCHG
179 EBF6 225D00            SHLD  :005D      Set addr next instr = 004C
180 EBF9 EB                XCHG
181 EBFA 7E                MOV   A,M        Get opcode of instr
182 EBFB E6C7              ANI   :C7
183 EBFD FEC7              CPI   :C7
184 EBFF CA33EC            JZ    :EC33      Jump if RST
185 EC02 3A5F00            LDA   :005F      Get int. mask
186 EC05 CD3EEF            CALL  :EF3E      Store it in TIC; set A=0
187 EC08 32F9FF            STA   :FFF9      Reset timer 1 (trap
```



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

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

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

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



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

```
                        * Stackpointer, TICC and GIC are not restored !
                        *
EC63 CDE7ED     L3E196  CALL  :EDE7     Exchange bytes in reg.
                                        save area
EC66 2A5300     L3E197  LHLD  :0053     Get stored PSW
EC69 E5                 PUSH  H
EC6A F1                 POP   PSW       Restore PSW
EC6B 2A5500             LHLD  :0055
EC6E 44                 MOV   B,H
EC6F 4D                 MOV   C,L       Restore BC
EC70 2A5700             LHLD  :0057
EC73 EB                 XCHG            Restore DE
EC74 2A5D00             LHLD  :005D
EC77 E5                 PUSH  H         Addr next instr on stack
EC78 2A5900             LHLD  :0059     Restore HL
EC7B C9                 RET             Goto addr in (005D/E)
                        *
                        **********************
                        * CALCULATE DE - HL *
                        **********************
                        *
                        * Exit: HL = DE - HL.
                        *       BCDE preserved, AF corrupted.
                        *
EC7C 7B         L3E198  MOV   A,E
EC7D 95                 SUB   L
EC7E 6F                 MOV   L,A       L=E-L
EC7F 7A                 MOV   A,D
EC80 9C                 SBB   H
EC81 67                 MOV   H,A       H=D-H-CY
EC82 C9                 RET
                        *
                        *************
                        * M - MOVE *
                        *************
                        *
                        * Moves a block of data (laddr - haddr) given to
                        * a given destination address (daddr).
                        *
                        * Exit: BC: 1st unused destination address.
                        *       DE: Last source address for direction of
                        *           movement.
                        *       HL: 1st unused source address.
                        *       AF: Corrupted.
                        *
EC83 0E03       MOVEK   MVI   C,:03     Nr of addr allowed
EC85 CDDEEA             CALL  :EADE     Get addr on stack
EC88 0D                 DCR   C         3 addr given ?
EC89 F262EA             JP    :EA62     Error if not
EC8C C1                 POP   B         daddr in BC
EC8D D1                 POP   D         haddr in DE
EC8E E1                 POP   H         laddr in HL
EC8F E5                 PUSH  H         Save laddr on stack
EC90 CD7CEC             CALL  :EC7C     Calc length of block to
                                        be moved
EC93 DA62EA             JC    :EA62     Error if wrong inputs
EC96 E3                 XTHL            laddr in HL, length on stack
EC97 79                 MOV   A,C       )
EC98 95                 SUB   L         ) Check if daddr < laddr
EC99 78                 MOV   A,B       )
EC9A 9C                 SBB   H         )
EC9B DAAEEC             JC    :ECAE     Then jump
```



```
                        * If daddr > laddr:
EC9E E3                 XTHL            length in HL, laddr on stack
EC9F 09                 DAD   B         daddr of highest byte
ECA0 44                 MOV   B,H       ) Store it in BC
ECA1 4D                 MOV   C,L       )
ECA2 E1                 POP   H         laddr in HL
ECA3 EB                 XCHG            DE: laddr, HL: haddr
ECA4 7E         L3E200  MOV   A,M       Get byte to be moved
ECA5 02                 STAX  B         Move it
ECA6 0B                 DCX   B         Decr pntr daddr
ECA7 CD58EC             CALL  :EC58     Check if ready
ECAA D8                 RC              Then quit
ECAB C3A4EC             JMP   :ECA4     Next byte
                        * If daddr < laddr:
ECAE E3         L3E201  XTHL            Length in HL, laddr on stack
ECAF E1                 POP   H         laddr in HL
ECB0 7E         L3E202  MOV   A,M       Get byte
ECB1 02                 STAX  B         And move it
ECB2 03                 INX   B         Incr pntr daddr
ECB3 CD80ED             CALL  :ED80     INX H; check if ready
ECB6 D8                 RC              Then quit
ECB7 C3B0EC             JMP   :ECB0     Next byte
                        *
                        *************
                        * Z - RESET *
                        *************
                        *
                        * Z1: Reset CPU save area 0051-005E. Initialise
                        *     stackpointer to F900 and save it in 005B/C.
                        *
                        * Z2: Sets: current interrupt mask (005F) = #C5,
                        *           TICC control word (0060) = #FC,
                        *           GIC  control word (0061) = #1B.
                        *     Initialises interrupt vector area #0062-#0071.
                        *     Sets interrupt vector RST 0 to #EB5D.
                        *
                        * Z3: Z1 + Z2.
                        *
                        * Exit: All registers corrupted.
                        *
ECBA 0E01       ZEROK   MVI   C,:01     Nr of databytes allowed
ECBC CDDEEA             CALL  :EADE     Get hexnr and store it
                                        on stack
ECBF E1                 POP   H         Get hexnr from stack
ECC0 7D                 MOV   A,L       into A
ECC1 F5                 PUSH  PSW       Save it again
ECC2 E602               ANI   :02
ECC4 CAE9EC             JZ    :ECE9     Jump if Z1 only
                        * If Z2 or Z3:
ECC7 3EC5               MVI   A,:C5     Set interrupt mask for
                                        clock, keyb, ext, timer 1
ECC9 325F00             STA   :005F     Preserve int.mask
ECCC 3EF4               MVI   A,:F4
ECCE 00                 NOP
ECCF 00                 NOP
ECD0 00                 NOP
```

```
188 ECD1 F608              ORI   :08
189 ECD3 216000            LXI   H,:0060
190 ECD6 77                MOV   M,A        Set TICC contr.word = FC
191 ECD7 23                INX   H
192 ECD8 361B              MVI   M,:1B      Set GIC contr.word = 1B
193 ECDA CD1ED9            CALL  :D91E      Init int.vector addr
194 ECDD 215DEB            LXI   H,:EB5D
195 ECE0 226200            SHLD  :0062      Set RST0 vector = EB5D
196 ECE3 00                NOP
197 ECE4 00                NOP
198 ECE5 00                NOP
199 ECE6 00                NOP
200 ECE7 00                NOP
201 ECE8 00          L3E204 NOP
202 ECE9 F1          ZERO1  POP   PSW        Get nr Z instr back
203 ECEA 1F                 RAR              Check for Z2 only
204 ECEB D0                 RNC              Ready if Z2 only
205
206                  * If Z1 or Z3:
207
208 ECEC 0E00               MVI   C,:00      )
209 ECEE 115E00             LXI   D,:005E    ) Init reg. save area:
210 ECF1 215100             LXI   H,:0051    )
211 ECF4 CD54ED             CALL  :ED54      ) Fill (0051-005E) with 00
212 ECF7 2100F9             LXI   H,:F900
213 ECFA F9                 SPHL             Set SP=F900
214 ECFB 225B00             SHLD  :005B      Save SP
215 ECFE C342EA             JMP   :EA42      Return for new inputs
216                  *
217                  ****************
218                  * PRINT SPACE *
219                  ****************
220                  *
221                  * Exit: AFC corrupted, BDEHL preserved.
222                  *
223                  TSP
224 ED01 0E20       LSP    MVI   C,:20
225 ED03 C3B4EE             JMP   :EEB4      Print space
226                  *
227                  ************************************
228                  * SCAN KEYBOARD, PRINT CHARACTER *
229                  ************************************
230                  *
231                  * Gets a keyboard input. Zeroes are ignored.
232                  * The character is printed (if not 'ESC').
233                  *
234                  * Exit: A : Character typed in.
235                  *       Other registers preserved.
236                  *
237 ED06 CDB8EE       CIE    CALL  :EEB8      Scan keyboard
238 ED09 E67F               ANI   :7F        Skip bit 7
239 ED0B CA06ED             JZ    :ED06      Ignore zeroes (await an
240                                           input)
241 ED0E FE12               CPI   :12        'ESC' ?
242 ED10 C5                 PUSH  B
243 ED11 4F                 MOV   C,A        Char in C
244 ED12 C4B4EE             CNZ   :EEB4      Print char if not 'ESC'
245 ED15 79                 MOV   A,C        Char in A
246 ED16 C1                 POP   B
247 ED17 C9                 RET
248                  *
249                  *
```



```
250                  ****************************
251                  * PRINT ADDRESS IN ASCII *
252                  ****************************
253                  *
254                  * LADDR: An address in HL is converted to ASCII and
255                  *        printed (4 nibbles).
256                  * LBYTE: A 1-byte value is printed in ASCII.
257                  *
258                  * Entry: LADDR: Address in HL.
259                  *        LBYTE: Value in A.
260                  * Exit:  AFC corrupted, BDEHL preserved.
261                  *
262 ED18 7C           LADDR  MOV   A,H        Hibyte in A.
263 ED19 CD1DED             CALL  :ED1D      Print both nibbles in ASCII
264 ED1C 7D                 MOV   A,L        Lobyte in A
265 ED1D F5          LBYTE  PUSH  PSW        Preserve hex value 2 char
266 ED1E 07                 RLC              )
267 ED1F 07                 RLC              ) Move hinibble
268 ED20 07                 RLC              ) into lonibble
269 ED21 07                 RLC              )
270 ED22 CD26ED             CALL  :ED26      Print lonibble
271 ED25 F1                 POP   PSW        Restore byte
272 ED26 E60F        L3E210 ANI   :0F        Lonibble only
273 ED28 CD40ED             CALL  :ED40      Convert it to ASCII
274 ED2B 4F                 MOV   C,A        And store it in C
275 ED2C C3B4EE             JMP   :EEB4      Print char in C
276                  *
277                  ****************
278                  * PRINT STRING *
279                  ****************
280                  *
281                  * Entry: HL points to string. End of string is 00.
282                  * Exit:  HL points to 00 at end of string.
283                  *        AFC corrupted, BDE preserved.
284                  *
285 ED2F 7E           L3E211 MOV   A,M        Get byte from string
286 ED30 4F                 MOV   C,A        into C
287 ED31 B7                 ORA   A          Byte = 00 ?
288 ED32 C8                 RZ               Then ready
289 ED33 CDB4EE             CALL  :EEB4      Print char in C
290 ED36 23                 INX   H          Pnts to next char
291 ED37 C32FED             JMP   :ED2F      Print next char
292                  *
293                  ************************
294                  * PRINT CARRIAGE RETURN *
295                  ************************
296                  *
297                  * Exit: AFC corrupted, BDEHL preserved.
298                  *
299 ED3A 0E0D         LCRLF  MVI   C,:0D
300 ED3C CDB4EE             CALL  :EEB4      Print 'CR'
301 ED3F C9                 RET
302                  *
303                  ********************************
304                  * CONVERT HEX NIBBLE TO ASCII *
305                  ********************************
306                  *
307                  * Entry: Hex value in A.
308                  * Exit:  ASCII value in A.
309                  *        BCDEHL preserved. F corrupted.
310                  *
311 ED40 C630         L3E213 ADI   :30        Convert
```

```
ED42 FE3A              CPI   :3A       Number 0-9?
ED44 D8                RC             Then ready
ED45 C607              ADI   :07       Convert A-F
ED47 C9                RET
                       *
                       ************
                       * F - FILL *
                       ************
                       *
                       * Fills a memory area between given boundaries
                       * with given data.
                       *
                       * Exit: CY=1. All registers corrupted.
                       *
ED48 0E03       FILLK  MVI   C,:03     Nr of addr/data allowed
ED4A CDDEEA            CALL  :EADE     Get addr/data on stack
ED4D 0D                DCR   C         3 blocks given?
ED4E F262EA            JP    :EA62     Error if not
ED51 C1                POP   B         Data in C
ED52 D1                POP   D         haddr in DE
ED53 E1                POP   H         laddr in HL
ED54 71         FILL   MOV   M,C       Data into memory
ED55 CD80ED            CALL  :ED80     INX H; check if ready
ED58 D8                RC             Then quit
ED59 C354ED            JMP   :ED54     Fill next addr
                       *
                       ******************
                       * S - SUBSTITUTE *
                       ******************
                       *
ED5C 0E01       SUBSK  MVI   C,:01     Nr of addr allowed
ED5E CDDEEA            CALL  :EADE     Get addr on stack
ED61 E1                POP   H         Addr in HL
ED62 7E                MOV   A,M       Get contents of addr
ED63 CD1DED            CALL  :ED1D     Print it in ASCII
ED66 AF                XRA   A
ED67 CD6AEE            CALL  :EE6A     Evt. modify contents
ED6A D262ED            JNC   :ED62     Next addr
ED6D C9                RET
                       *
                       ***************
                       * X - EXAMINE *
                       ***************
                       *
ED6E 219DEE     EXAMK  LXI   H,:EE9D   Startaddr register table
ED71 115300            LXI   D,:0053   Startaddr CPU save area
ED74 C339EE            JMP   :EE39     Go to display routine
                       *
                       **********************
                       * V - VECTOR EXAMINE *
                       **********************
                       *
ED77 21A8EE     VECXK  LXI   H,:EEA8   Startaddr vector table
ED7A 115F00            LXI   D,:005F   Startaddr vector area
ED7D C339EE            JMP   :EE39     Go to display routine
                       *
                       ************************************
                       * INCREMENT HL AND COMPARE WITH DE *
                       ************************************
                       *
                       * Exit: HL was FFFF: CY=1, Z=1.
                       *       Else: New HL < DE : CY=0.
```



```
                       *             New HL = DE :  Z=1.
                       *             New HL > DE : CY=1.
                       *       BCDE preserved, HL=HL+1, AF corrupted.
                       *
ED80 23         INXCK  INX   H
ED81 37                STC             CY=1
ED82 7C                MOV   A,H
ED83 B5                ORA   L
ED84 C8                RZ              Abort if new HL is 0000
ED85 7B                MOV   A,E
ED86 95                SUB   L
ED87 7A                MOV   A,D
ED88 9C                SBB   H
ED89 C9                RET
                       *
                       **********
                       * G - GO *
                       **********
                       *
                       * Reads one field if given.
                       * If no field given: Restores CPU registers, goes
                       *    to PC address. Returnaddress is #EA42 (into
                       *    command loop).
                       * If field given: Saves it as PC, initialises TICC
                       *    and GIC. Returnaddress is #EA04. Goes to the
                       *    given address.
                       * REMARK: The stackpointer is never restored !
                       *
ED8A 0E01       GOK    MVI   C,:01
ED8C CD07EB            CALL  :EB07     Scan keyb; addr on stack
ED8F 0D                DCR   C         No addr given?
ED90 F5                PUSH  PSW
ED91 CC56EB            CZ    :EB56     Then check if 'CR'
ED94 F1                POP   PSW
ED95 CA63EC            JZ    :EC63     No addr: restore CPU reg
                                       (but not SP/TICC/GIC !) and
                                       go to addr in 005D/E
ED98 E1                POP   H         'GO' addr in HL
ED99 225D00            SHLD  :005D     Save it
ED9C 2A6000     L3E221 LHLD  :0060     Get GIC/TICC init values
ED9F 7C                MOV   A,H       GIC init value in A
EDA0 CDD5ED            CALL  :EDD5     Init GIC
EDA3 7D                MOV   A,L       TICC init value in A
EDA4 CDB7ED            CALL  :EDB7     Init TICC
EDA7 3A5F00            LDA   :005F     Get current int mask
EDAA 32F8FF            STA   :FFF8     Set int mask
EDAD CDE7ED            CALL  :EDE7     exch. bytes in 0051-5E
EDB0 2104EA            LXI   H,:EA04   Returnaddr for 'GO'
EDB3 E5                PUSH  H         Save EA04 on stack
EDB4 C366EC            JMP   :EC66     Restore CPU reg and 'GO'
                       *
                       * INITIALISE TICC:
                       *
                       * Entry: A: Initial value TICC command word (#FC).
                       * Exit:  AFBC corrupted, DEHL preserved.
                       *
EDB7 47         L3E222 MOV   B,A       Init.value in B
EDB8 07                RLC            A=F9
EDB9 F5                PUSH  PSW      On stack: A=F9, CY=1
EDBA 07                RLC            A=F3
EDBB 07                RLC            A=E7
EDBC 07                RLC            A=CF
```

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



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

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

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

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



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

```
EE63 C8              RZ              Quit if ready
EE64 CD01ED          CALL  :ED01     Print space
EE67 C344EE          JMP   :EE44     Next one
                *
                ******************************
                * V+X: EVT. MODIFY CONTENTS *
                ******************************
                *
                * Entry: HL: Memory address.
                *         A : Symbol.
                *
EE6A B7         L3E326  ORA   A         Set flags on symbol
EE6B F5                 PUSH  PSW       and save it
EE6C 0E2D               MVI   C,:2D
EE6E CDB4EE             CALL  :EEB4     Print '-'
EE71 E5                 PUSH  H         Save addr mem.area
EE72 0E01               MVI   C,:01     Nr of datablocks allowed
EE74 CD07EB             CALL  :EB07     Get input on stack; last
                                        byte typed in in B
EE77 0D                 DCR   C
EE78 CA87EE             JZ    :EE87     Jump if incorrect input
EE7B D1                 POP   D         Data typed in in DE
EE7C E1                 POP   H         Get addr mem.area
EE7D F1                 POP   PSW       Get symbol and flags
EE7E 73                 MOV   M,E       Change memory contents
EE7F F28DEE             JP    :EE8D     Jump if '1 byte' symbol

                * If 2-byte symbol:

EE82 23                 INX   H
EE83 72                 MOV   M,D       Change 2nd byte
EE84 C38DEE             JMP   :EE8D
                *
EE87 E1         L3E327  POP   H         Retrieve addr mem.area
EE88 F1                 POP   PSW       Retrieve symbol + flags
EE89 F28DEE             JP    :EE8D     If '1 byte' symbol
EE8C 23                 INX   H         Add. INX H if 2-byte symbol
EE8D 23         L3E328  INX   H         Next mem.area
EE8E 78                 MOV   A,B       Get last input
EE8F FE0D               CPI   :0D
EE91 C8                 RZ              Ready if 'CR'
EE92 FE20               CPI   :20
EE94 C8                 RZ              Ready if 'SP'
EE95 FE12               CPI   :12
EE97 37                 STC             Abort with CY=1 if 'ESC'
EE98 C8                 RZ
EE99 C362EA             JMP   :EA62     Else: wrong input: Error
                *
                ******************************
                * SYMBOL TABLE EXAMINE (X) *
                ******************************
                *
                * The msb is '1' for symbols of two-byte
                * registers.
                *
EE9C C9         L3E405  DATA  :C9       I (addr current instr)
EE9D 41                 DATA  :41       A
EE9E 46                 DATA  :46       F (flags)
EE9F 42                 DATA  :42       B
EEA0 43                 DATA  :43       C
EEA1 44                 DATA  :44       D
EEA2 45                 DATA  :45       E
```



```
EEA3 48                 DATA  :48       H
EEA4 4C                 DATA  :4C       L
EEA5 D3                 DATA  :D3       S (stackpointer)
EEA6 D0                 DATA  :D0       P (program counter)
EEA7 00                 DATA  :00
                *
                ********************************
                * SYMBOL TABLE VECTOR EXAMINE *
                ********************************
                *
                * The msb is '1' for symbols of two-byte
                * registers.
                *
EEA8 4D         L3E406  DATA  :4D       M (TICC int mask)
EEA9 54                 DATA  :54       T (TICC cmd + comm.reg)
EEAA 47                 DATA  :47       G (GIC cmd word)
EEAB B0                 DATA  :B0       0  )
EEAC B1                 DATA  :B1       1  )
EEAD B2                 DATA  :B2       2  )
EEAE B3                 DATA  :B3       3  ) Interrupt vectors
EEAF B4                 DATA  :B4       4  ) addresses
EEB0 B5                 DATA  :B5       5  )
EEB1 B6                 DATA  :B6       6  )
EEB2 B7                 DATA  :B7       7  )
EEB3 00                 DATA  :00
                *
                ********************
                * PRINT CHARACTER *
                ********************
                *
                * Entry: C: Character to be printed.
                * Exit:  AF corrupted, BCDEHL preserved.
                *
EEB4 79         CO      MOV   A,C       Char in A
EEB5 C360DD             JMP   :DD60     Print char
                *
                ********************************
                * SCAN KEYBOARD; IGNORE BREAK *
                ********************************
                *
                * Exit: Character in A.
                *       BCDEHL preserved.
                *
EEB8 CD83EF     CI      CALL  :EF83     Scan keyboard
EEBB DAB8EE             JC    :EEB8     Ignore break
EEBE CAB8EE             JZ    :EEB8     Wait for input
EEC1 C9                 RET
                *
                ************************************
                * SCAN KEYBOARD FOR BREAK PRESSED *
                ************************************
                *
                * Exit: CY=0: No break.
                *       CY=1: Break pressed.
                *       A corrupted, BCDEHL preserved.
                *
EEC2 CDA5D6     BREAK   CALL  :D6A5     Scan keyb for new inputs
EEC5 D0                 RNC             abort if Break not pressed
EEC6 C36AEA             JMP   :EA6A     Restore SP, wait for
                                        new inputs
                *
                *
```

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



```
EF0E C9                 RET
                        *
                        ************
                        * R - READ *
                        ************
                        *
                        * One address is allowed. An evt. name is stored
                        * in the EBUF. Reads header, startaddress and data
                        * from tape. Errorcheck only on data.
                        *
                        * Exit: HL: 1st address above file loaded.
                        *       BC: Evt. offset.
                        *       AFDE corrupted.
                        *
EF0F 0E01       RHEXK   MVI   C,:01     Nr of addr allowed
EF11 CDDEEA             CALL  :EADE     Scan keyb; addr on stack
EF14 CD48EF             CALL  :EF48     Evt name in input buffer
EF17 0631               MVI   B,:31     File type byte
EF19 00                 NOP
EF1A 00                 NOP
EF1B 00                 NOP
EF1C 0E00               MVI   C,:00
EF1E CDD8EE             CALL  :EED8     Read file header
EF21 1100F9             LXI   D,:F900   Max addr to write data into
EF24 C1                 POP   B         Get evt offset from stack
EF25 CD74EF             CALL  :EF74     Read startaddr from tape
EF28 09                 DAD   B         Add offset
EF29 CD8AEF             CALL  :EF8A     Read data block + trailer
EF2C D262EA             JNC   :EA62     Print 'error' if reading
                                        error
EF2F C9                 RET
                        *
                        ****************************************
                        * RST 0: POINTER TO 'LOOK'-FLAG IN HL *
                        ****************************************
                        *
                        * Part of 3EC21.
                        *
                        * Exit: A=0, BCDE preserved.
                        *
EF30 AF         L3E343  XRA   A
EF31 215000             LXI   H,:0050
EF34 C9                 RET
                        *
                        ********************
                        * INITIALISE RST 0 *
                        ********************
                        *
                        * Entry: On stack: Returnaddress #EA42.
                        *
EF35 3EFB       L3E344  MVI   A,:FB     Instr code for EI in A
EF37 324700             STA   :0047     Save it
EF3A D1                 POP   D         Get EA42 in DE
EF3B C3E2EB             JMP   :EBE2     Into RST 0
                        *
                        **********************
                        * SET INTERRUPT MASK *
                        **********************
                        *
                        * Part of RST0 (3EC05).
                        *
                        * Entry: A: Value for TICC interrupt mask.
```

```
                        * Exit:  A=0, F corrupted, BCDEHL preserved.
                        *
EF3E 32F8FF     L3E345  STA   :FFF8     Set TICC int mask
EF41 AF                 XRA   A         A=0
EF42 C9                 RET
                        *
EF43 FF                 DATA  :FF
                        *
                        ***************************
                        * part of RST 0 (3EC3B) *
                        ***************************
                        *
EF44 2B         L3E346  DCX   H
EF45 C319EC             JMP   :EC19     Into RST 0
                        *
                        ******************************
                        * W+R: NAME IN INPUT BUFFER *
                        ******************************
                        *
                        * Names >126 characters destroy BASIC pointers.
                        *
                        * Entry: Last character typed in, in B.
                        * Exit:  BCD preserved. HL points to EBUF.
                        *        A= 0: No file name given.
                        *        A<>0: File name given.
                        *
EF48 213E01     L3E347  LXI   H,:013E   Startaddr EBUF
EF4B 1EFF               MVI   E,:FF
EF4D 78                 MOV   A,B       Last char in A
EF4E D60D               SUI   :0D       'CR' ?
EF50 77                 MOV   M,A       (13E) is 0 if CR
EF51 C8                 RZ              Quit if no name given

                        * If name given:

EF52 E5                 PUSH  H         Save startaddr EBUF
EF53 2C         L3E356  INR   L         Points to next loc
EF54 1C                 INR   E         Calc length
EF55 CD06ED             CALL  :ED06     Scan keyb, print char
EF58 77                 MOV   M,A       Char into EBUF
EF59 FE0D               CPI   :0D
EF5B C253EF             JNZ   :EF53     Next char if not 'CR'
EF5E E1                 POP   H         Retrieve startaddr EBUF
EF5F 73                 MOV   M,E       Store length name in 013E
EF60 C9                 RET
                        *
                        ***************
                        * (Not used) *
                        ***************
                        *
EF61 E1         L3E348  POP   H
EF62 C9                 RET
                        *
                        ********************************
                        * WRITE STARTADDRESS ON TAPE *
                        ********************************
                        *
                        * Entry: HL: Startaddress.
                        * Exit:  AF corrupted, BCDEHL preserved.
                        *
EF63 D5         L3E349  PUSH  D
EF64 E5                 PUSH  H
```



```
EF65 223E01             SHLD  :013E     Startaddr in EBUF
EF68 213E01             LXI   H,:013E   Startaddr to write from
EF6B 110200             LXI   D,:0002   Length
EF6E CDCFEE             CALL  :EECF     Write addr on tape
EF71 E1                 POP   H
EF72 D1                 POP   D
EF73 C9                 RET
                        *
                        ********************************
                        * READ STARTADDRESS FROM TAPE *
                        ********************************
                        *
                        * Exit: HL: Startaddress.
                        *       CY=1: No reading error.
                        *       CY=0: Reading error, errorcode in A.
                        *       BCDE preserved.
                        *
EF74 D5         L3E350  PUSH  D
EF75 213E01             LXI   H,:013E   Addr EBUF
EF78 114101             LXI   D,:0141   Addr after addr in EBUF
EF7B CDDEEE             CALL  :EEDE     Read block from tape
EF7E D1                 POP   D
EF7F 2A3E01             LHLD  :013E     Startaddr in HL
EF82 C9                 RET
                        *
                        *****************
                        * SCAN KEYBOARD *
                        *****************
                        *
                        * Part of 3EEB8. Scans keyboard. Returns
                        * any key received.
                        *
                        * Exit: A: Key received.
                        *       BCDEHL preserved.
                        *       CY=1: Break pressed.
                        *
EF83 AF         L3E351  XRA   A
EF84 32B902             STA   :02B9     Enable complete keyb scan
EF87 C3BED6             JMP   :D6BE     Scan keyboard
                        *
                        ************************
                        * READ DATA FROM TAPE *
                        ************************
                        *
                        * Part of READ (3EF29).
                        *
EF8A CDDEEE     L3E352  CALL  :EEDE     Read block from tape
EF8D C3E1EE             JMP   :EEE1     Stop reading
                        *
                        ******************************
                        * DCE INITIALISATION ROUTINE *
                        ******************************
                        *
                        * Part of RESET (C719). Bootstrap for disc drive.
                        * Sets GIC in initialisation status. Checks if any
                        * input is received from the DCE-bus and performs
                        * the received instructions.
                        *
                        * Exit: A=#EE if no DCE-inputs available.
                        *
EF90 3E98       L3E353  MVI   A,:98
EF92 3203FE             STA   :FE03     PA+PCH in, PB+PCL out
```

```
498 EF95 3E07              MVI   A,:07
499 EF97 3203FE            STA   :FE03     PC3=1 ->
500 EF9A 3E01              MVI   A,:01
501 EF9C 3201FE            STA   :FE01     Output PB: 01
502 EF9F 3E01              MVI   A,:01
503 EFA1 3203FE            STA   :FE03     PC0=1
504 EFA4 010010            LXI   B,:1000
505 EFA7 3A02FE    L3E355  LDA   :FE02     Get input from PCH
506 EFAA E620              ANI   :20       Bit 5 only
507 EFAC C2B8EF            JNZ   :EFB8     Jump if inputs received
508                                                    -> IFBit5=1
509                * If no inputs:
510
511 EFAF 0B                DCX   B         )
512 EFB0 78                MOV   A,B       ) Wait loop until C=#10
513 EFB1 B1                ORA   C         )
514 EFB2 C2A7EF            JNZ   :EFA7     )
515 EFB5 3EEE              MVI   A,:EE     A=EE if no inputs
516 EFB7 C9                RET
517                *
518                * DCE BOOTSTRAP INPUT ROUTINE:
519                *
520                * Loads MLP inputs from the DCE-bus into the
521                * stackbottom and goes to it.
522                *
523 EFB8 1100F8    L3E354  LXI   D,:F800   Addr stackbottom
524 EFBB 3E05      L3E357  MVI   A,:05
525 EFBD 3203FE            STA   :FE03     PC2=1
526 EFC0 3A02FE    L3E358  LDA   :FE02     Get input from PC
527 EFC3 E680              ANI   :80       Bit 7 only
528 EFC5 CAC0EF            JZ    :EFC0     Wait for change to high
529 EFC8 3A00FE            LDA   :FE00     Get input from PA
530 EFCB 12                STAX  D         Save input in stack area
531 EFCC 13                INX   D         Point to next loc
532 EFCD 3E04              MVI   A,:04
533 EFCF 3203FE            STA   :FE03     PC2=0
534 EFD2 3A02FE    L3E359  LDA   :FE02     Get input from PC
535 EFD5 E680              ANI   :80       Bit 7 only
536 EFD7 C2D2EF            JNZ   :EFD2     Wait for change to low
537 EFDA 3A02FE            LDA   :FE02     Get input from PC
538 EFDD E620              ANI   :20       Bit 5 only
539 EFDF C2BBEF            JNZ   :EFBB     Again if high
540 EFE2 3E06              MVI   A,:06
541 EFE4 323EFE            STA   :FE3E     (FE3E hardwarewise read as
542                                        FE02). PC=06
543 EFE7 3A02FE    L3E360  LDA   :FE02     Get input from PC
544 EFEA E620              ANI   :20       Bit 5 only
545 EFEC CAE7EF            JZ    :EFE7     Wait for change to high
546 EFEF C300F8            JMP   :F800     Go to stackbottom
547                *
548 EFF2 FF                DATA  :FF
549 EFF3 FF                DATA  :FF
550                *
551                **********************
552                * SCAN 'DINC' INPUT *
553                **********************
554                *
555                * Part of 3E935. Default 'DINC' is RS232 input.
556                *
557 EFF4 CDB4DD    L3E361  CALL  :DDB4     Get input from DINC
558 EFF7 CA35E9            JZ    :E935     Scan keyb if no DINC input
559
```



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

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

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

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



```
064                 ************************
065                 * START FROM SCRATCH *
066                 ************************
067                 *
068                 * Added is the reset of all keyboard pointers at
069                 * various moments:
070                 * - When restarting the Basic interpreter.
071                 * - After command execution. Then also the output
072                 *   direction is reset to screen (to avoid a not
073                 *   useable keyboard).
074                 * - Idem before executing 'break'.
075                 * Resetting the keyboard pointers clears the key
076                 * input buffer in order to avoid keybounce.
077                 *
078                         ORG   :C827
079 C827 CD63D5            CALL  :D563     ] Keyb. pntrs to default
080 C82A 210000            LXI   H,:0000
081 C82D 220001            SHLD  :0100
082 C830 220401            SHLD  :0104
083 C833 221301            SHLD  :0113
084 C836 7C                MOV   A,H
085 C837 322201            STA   :0122
086                 *
087                         ORG   :C897
088 C897 21C402            LXI   H,:02C4   ]
089 C89A 7E                MOV   A,M       ] Get break flag
090 C89B B7                ORA   A
091 C89C CA7FC8            JZ    :C87F
092 C89F CD63D5            CALL  :D563     ] Keyb.pntrs to default
093 C8A2 AF                XRA   A         ]
094 C8A3 323101            STA   :0131     ] Output to screen
095                 *
096                         ORG   :C8AC
097 C8AC CA9FC8            JZ    :C89F     ] Keyb.pntrs to default,
098                                         output to screen
099                 *
100                 **************************
101                 * EMERGENCY STOP ROUTINE *
102                 **************************
103                 *
104                 * No change in execution, only using a new entry on
105                 * 'Run NEW'.
106                 *
107                         ORG   :CA25
108 CA25 CDB0DE            CALL  :DEB0     Run NEW with default heap
109                 *
110                 ***********
111                 * RUN NEW *
112                 ***********
113                 *
114                 * - An additional entry is made for run NEW with
115                 *   returning the heapsize to its default value.
116                 *   In BASIC V1.0, this entry was available on
117                 *   #CEC6.
118                 * - All jumps/calls to RNEW with other heapsizes
119                 *   are updated.
120                 *
121                         ORG   :DEB0
122 DEB0 210001            LXI   H,:0100   ]
123 DEB3 229D02            SHLD  :029D     ] Set heap to default size
124 DEB6 00                NOP
125 DEB7 00                NOP             Into RNEW
```

```
                        *
                                ORG   :C778
C778 CDB8DE                     CALL  :DEB8     !
                        *
                                ORG   :CF02
CF02 B8DE                       DBL   :DEB8     !
                        *
                                ORG   :D2A9
D2A9 CDB8DE                     CALL  :DEB8     !
                        *
                        ************
                        * RUN SAVE *
                        ************
                        *
                        * Instructions only moved.
                        *
                                ORG   :D257
D257 3E30                       MVI   A,:30
D259 CDC502                     CALL  :02C5
D25C E1                         POP   H
D25D D1                         POP   D
D25E CDC802                     CALL  :02C8
D261 D1                         POP   D
D262 C3D8D7                     JMP   :D7D8
                        *
                        ************
                        * RUN LOAD *
                        ************
                        *
                        * Instructions only moved.
                        *
                                ORG   :D274
D274 CD23CB                     CALL  :CB23
D277 CD91E7                     CALL  :E791
D27A C5                         PUSH  B
                        *
                                ORG   :CF16
CF16 74D2                       DBL   :D274     !
                        *
                        ************
                        * RUN CHECK *
                        ************
                        *
                        * CALL :D7E6 is replaced by the contents of D7E6.
                        * No change in execution.
                        *
                                ORG   :D2C9
D2C9 CDCE02                     CALL  :02CE
D2CC FE33                       CPI   :33
D2CE D2EBD2                     JNC   :D2EB
D2D1 0C                         INR   C
D2D2 3E00                       MVI   A,:00     ] Old contents
D2D4 CDD702                     CALL  :02D7     ] D7E6
D2D7 00                         NOP
                        *
                        ****************************
                        * PRINT MESSAGE ON NEW LINE *
                        ****************************
                        *
                        * New routine. It moves the cursor to a new line
                        * before a message is printed. Used for printing
                        * 'STOPPED IN LINE ...' and 'END PROGRAM'.
```



```
                        *
                                ORG   :D7E6
D7E6 CD55DD                     CALL  :DD55     ] Cursor to column 0
D7E9 C3FFDA                     JMP   :DAFF     ] Print message

                        * RSTOP:

                                ORG   :DF03
DF03 CDE6D7                     CALL  :D7E6     ]

                        * REND:

                                ORG   :DF0C
DF0C CDE6D7                     CALL  :D7E6     ]
                        *
                        ******************
                        * FAST PRINTING *
                        ******************
                        *
                        * Instruction only moved.
                        *
                                ORG   :D7EC
D7EC E5                         PUSH  H
                        *
                                ORG   :D3CC
D3CC C4ECD7                     CNZ   :D7EC     !
                        *
                                ORG   :D78D
D78D C3ECD7                     JMP   :D7EC     !
                        *
                        ************************
                        * READ BLOCK FROM TAPE *
                        ************************
                        *
                        * Now cassette motors are switched off directly
                        * after reading is finished.
                        * Especially in LOADA, switching off the cassette
                        * motors after re-arranging all data was very
                        * annoying.
                        *
                                ORG   :D89A
D89A C365D2                     JMP   :D265     ]
                        *
                                ORG   :D265
D265 D2B3D2                     JNC   :D2B3     ] Evt run loading error
D268 C3D402                     JMP   :02D4     ] Stop cassette motors
                        *
                        ********************
                        * INPUT TEXT LINE *
                        ********************
                        *
                        * Added: Output back to screen if break pressed.
                        * This avoids a not-useable keyboard on return.
                        *
                                ORG   :DD2D
DD2D DA36CD                     JC    :CD36     ] Jump if break pressed
                        *
                                ORG   :CD36
CD36 AF                         XRA   A         ] A=0
CD37 C35AD5                     JMP   :D55A     ]
CD3A FF                         DATA  :FF       ]
CD3B FF                         DATA  :FF       ]
```

```
                        *
                                ORG   :D55A
D55A 323101                     STA   :0131     ] Output to screen
D55D C349DD                     JMP   :DD49     ] Ignore line if break
                                              pressed
                        *
                        ********************
                        * MULTIPLY HL * A *
                        ********************
                        *
                        * Routine only moved.
                        *
                                ORG   :DE9C
DE9C DAA9DE                     JC    :DEA9     !
DE9F B7                         ORA   A
DEA0 CAACDE                     JZ    :DEAC     !
DEA3 EB                         XCHG
DEA4 29                         DAD   H
DEA5 EB                         XCHG
DEA6 D296DE                     JNC   :DE96
DEA9 D1                         POP   D
DEAA F1                         POP   PSW
DEAB C9                         RET
DEAC D1                         POP   D
DEAD F1                         POP   PSW
DEAE 3F                         CMC
DEAF C9                         RET
                        *
                        ************
                        * RUN RUN *
                        ************
                        *
                        * The sequence of the instructions is modified.
                        * This enables RUN <linenr> to be used without
                        * destroying the contents of heap and symtab.
                        *
                                ORG   :DF9E
DF9E CD23CB                     CALL  :CB23     Empty heap and symtab
DFA1 2A9F02                     LHLD  :029F     Get start textbuf
DFA4 44                         MOV   B,H       ) in BC
DFA5 4D                         MOV   C,L       )
DFA6 CD01E4                     CALL  :E401     Run RESTORE
DFA9 210000                     LXI   H,:0000
DFAC 221501                     SHLD  :0115     Reset step/trace flag
DFAF AF                         XRA   A
DFB0 322601                     STA   :0126     No suspended program
DFB3 3100F9                     LXI   SP,:F900  Reset SP

                        * RUN <linenr> :

                                ORG   :DFBD
DFBD C3A4DF                     JMP   :DFA4     ! Process RUN
                        *
                        ****************************
                        * STORE DATA (READ, INPUT) *
                        ****************************
                        *
                        * Handle string inputs (no change):
                        *
                                ORG   :E385
E385 CCB8E4                     CZ    :E4B8     ] Only JZ :E3A2 replaced
E388 C4B4E4                     CNZ   :E4B4     ]       by its contents
```



```
                        * Error exit changed: Now checks if end of input
                        * is reached. EFEPT is only updated if not running
                        * inputs (anymore).
                        *
                                ORG   :E39C
E39C CA67E3                     JZ    :E367     ] Read next data if no error
E39F 3A1701                     LDA   :0117     If error: Get 'run-input'
                                              flag
E3A2 CDC3E3                     CALL  :E3C3     ]
E3A5 C30BDA                     JMP   :DA0B     ] Run 'SYNTAX ERROR'
                        *
                                ORG   :E3C3
E3C3 A7                         ANA   A         ] Set flags
E3C4 C0                         RNZ             ] Abort if running inputs
E3C5 2A3201                     LHLD  :0132     Get EFEPT
E3C8 11FCFF                     LXI   D,:FFFC
E3CB 19                         DAD   D         -4
E3CC 220001                     SHLD  :0100     Set start current line
E3CF C9                         RET
                        *
                        ****************************
                        * PREPARE GETTING INPUTS *
                        ****************************
                        *
                        * The keyboard pointers are set to their default
                        * values before inputs are asked. This avoids
                        * keybounce.
                        *
                                ORG   :E447
E447 221D01                     SHLD  :011D
E44A CD63D5                     CALL  :D563     ] Init keyb pntrs
E44D 3EFF                       MVI   A,:FF
E44F 321701                     STA   :0117
E452 C3D0E3                     JMP   :E3D0
                        *
                        ******************
                        * cont. of 0E401 *
                        ******************
                        *
                        * Instruction only moved.
                        *
E455 322301                     STA   :0123
E458 C9                         RET
E459 FF                         DATA  :FF
                        *
                                ORG   :E409
E409 C355E4                     JMP   :E455     !
                        *
                        *************
                        * RUN CLEAR *
                        *************
                        *
                        * Routine is completely modified.
                        * Max. useable heap space in V1.0 was #7FFF-4.
                        * Now it is #7FFF.
                        * Doesnot set up anymore a complete new heap, but
                        * just empties heap and symboltable entries and
                        * shifts the program to after the new heap.
                        *
                                ORG   :E6B5
E6B5 CDF8E6                     CALL  :E6F8     ] Get reqd space in HL
```

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



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

```
                        *
                                ORG   :EC71
                                CZ    :DACC     ] If to be waited
                                CNZ   :CD62     ] Goto MLP address
                                JMP   :CD67     Handle next code
EC71 CCCCDA                     DATA  :FF
EC74 C462CD
EC77 C367CD
EC7A FF

                        * If wait:
                        * As old routine, but HL replaced by DE.

                                ORG   :DACC
DACC 1B                         DCX   D         ]
DACD 7A                         MOV   A,D       ]
DACE B3                         ORA   E         ]
DACF C2CCDA                     JNZ   :DACC     ] Wait for DE=0
DAD2 C9                         RET             ]
DAD3 FF                         DATA  :FF       ]

                        * To enable ML call:

                                ORG   :CD62
CD62 D5                         PUSH  D         ] Address MLP on stack
CD63 C9                         RET             ] Go to it
                        *
                        ************************
                        * TEST A FPT VARIABLE *
                        ************************
                        *
                        * The old routine tested all bytes of the FPT nr.
                        * Now only the exponent byte and the hibyte of the
                        * mantissa is tested. In this way, very small FPT
                        * numbers are considered to be zero.
                        *
                                ORG   :EC91
EC91 CA98EC                     JZ    :EC98     ] Abort if A and B are 0
EC94 7B                         MOV   A,E       ] Exp byte in A
EC95 F601                       ORI   :01       ] Clear CY-flag
EC97 00                         NOP             ]
                        *
                        *
                        *    ============
                        ***** ROM BANK 1 *****
                        *    ============
                        *
                        *
                        ************
                        * FPT EXP *
                        ************
                        *
                        * Test for overflow is modified. Overflow occurs
                        * for e^X when -45 < X < 43.6.
                        * This is checked now before exponent routine is
                        * entered.
                        *
                                ORG   :E688
E688 CAC9EF                     JZ    :EFC9     ]
                        *
                                ORG   :EFC9
EFC9 7A                         MOV   A,D       ] Get lobyte MACC
EFCA E6C0                       ANI   :C0       ] Check for max. value
EFCC C28BE6                     JNZ   :E68B     ] Jump if nr too big
EFCF C394E6                     JMP   :E694     ] If O.K.
```



```
                        *
                        *********
                        * LOADA *
                        *********
                        *
                        * Switching off the cassette motors is now part of
                        * D897. So the cassette motors are switched off
                        * directy after the reading from tape is done.
                        *
                                ORG   :EE6B
EE6B C332EE                     JMP   :EE32     ] Adapted to new situation
                        *
                        *
                        *    ============
                        ***** ROM BANK 2 *****
                        *    ============
                        *
                        *
                        ***************
                        * WINDOW DOWN *
                        ***************
                        *
                        * A bug which let the cursor disappear sometimes is
                        * removed.
                        *
                                ORG   :ECB6
ECB6 2AA900                     LHLD  :00A9     ] Get offset top of window
                                                from start buffer
ECB9 EB                         XCHG            ] in DE
ECBA CD6BD2                     CALL  :D26B     ]
ECBD FCABED                     CM    :EDAB     Then cursor down
                        *
                                ORG   :D26B
D26B 2AAC00                     LHLD  :00AC     ] Get Y-offset cursor in
                                                document
D26E CDF2E6                     CALL  :E6F2     ] HL=HL-DE
D271 2D                         DCR   L         ] -1
D272 37                         STC             ] Set 'window changed' flag
D273 C9                         RET             ]
                        *
                        *
                        *    ============
                        ***** ROM BANK 3 *****
                        *    ============
                        *
                        *
                        ****************************************
                        * ERROR EXIT 'ENCODE INT NR INTO EBUF *
                        ****************************************
                        *
                                ORG   :E8B7
E8B7 219FE3                     LXI   H,:E39F   ] (0) Addr routine 'STORE
                                                DATA'
E8BA E5                         PUSH  H         ] Preserve it as returnaddr
E8BB 3A4000                     LDA   :0040     ] Get POROM
E8BE E63F                       ANI   :3F       ] Select bank 0
E8C0 F5                         PUSH  PSW       ]
E8C1 C3E6C6                     JMP   :C6E6     ] Bank return
                        *
                        *
                        *
                                END
E8C4
```