```
;-----------------------------------------------------------------------------------------------------------------------------------
;Table of contents
;
```

```
;Chapter 4 SOFTWARE ENVIRONMENT
;         4.1 I4004 instruction set summary
;         4.2 Using of ROM areas
;         4.3 Using of RAM areas
;         4.4 Basics of operation and implementation of the calculator
;         4.5 Square root implementation
;         4.6 Unusual results
;         4.7 Basic pseudo instruction codes
;         4.8 Square root pseudo instruction codes
;
;Chapter 5 Detailed analysis of the assembly code
;-----------------------------------------------------------------------------------------------------------------------------------

;-----------------------------------------------------------------------------------------------------------------------------------
;Chapter 1      INTRODUCTION
;
;This document is an analysis of Busicom 141-PF calculator built with Intel 4004, the world's first microprocessor. The software of
;the calculator can be considered the first program has ever made for microprocessors.
;-----------------------------------------------------------------------------------------------------------------------------------

;-----------------------------------------------------------------------------------------------------------------------------------
;Chapter 2      ABBREVIATIONS
;
;ACC            i4004 accumulator
;BPC            basic pseudo code
;CY             i4004 carry flag
;CR             constant register
;DP             digit point
;DR             dividend register
;IR             indirect register
;KR             keyboard buffer register
;M0..M15        main memory cells 0..15
;MR             memory register
;NR             number register
;R0..R15        i4004 register 0..15
;RR             result register
;QPC            square root pseudo code
;S0..S3         status character 0..3
;SQRT           square root
;SR             sub total register
;TR             main total register
;WR             working register
;-----------------------------------------------------------------------------------------------------------------------------------

;-----------------------------------------------------------------------------------------------------------------------------------
;Chapter 3 HARDWARE ENVIRONMENT:
;-----------------------------------------------------------------------------------------------------------------------------------

;-----------------------------------------------------------------------------------------------------------------------------------
;3.1    MCS-4 family components
;
;The Busicom calculator has been built with the Intel's MCS-4 family set using the following main components:
;       5 * i4001 256 * 8 bit ROM with 4 bit input/output port (5th is optional for square root function)
;       2 * i4002 320 bit RAM with 4 bit output port
;       3 * i4003 10 bit shift register
```

```
;        1 * i4004 central processor unit (CPU)
;
;The port bits of ROMs, RAMs and TEST pin of CPU are used for:
;
;TEST:  printer drum sector signal
;
;ROM0:  shifter output
;        bit0 = keyboard matrix column shifter clock (for a i4003 shifter)
;        bit1 = shifter data (shared for printer and keyboard matrix shifter)
;        bit2 = printer shifter clock (for two cascaded i4003 shifter)
;        bit3 = not used
;
;ROM1:  keyboard matrix rows input
;
;ROM2:  bit0 = printer drum index signal input
;        bit1 = not used
;        bit2 = not used
;        bit3 = printer paper advancing button input
;
;ROM3:  not used
;
;ROM4:  not used
;
;RAM0:  printer control outputs
;        bit0 = printing color (0=black, 1=red)
;        bit1 = fire print hammers
;        bit2 = not used
;        bit3 = advance the printer paper
;
;RAM1:  status light outputs
;        bit0 = memory lamp
;        bit1 = overflow lamp
;        bit2 = minus sign lamp
;        bit3 = not used
;
;According to the MCS-4 specification the clock period of the system should be between 1.35 and 2.00 microsec. In the calculator
;the minimum value is applied, which gives 10.8 microsec instruction cycle time for the simplest instructions and 21.6 microsec
;for the two cycle instructions; and the CPU runs nearly at 740 kHz clock speed.
;-------------------------------------------------------------------------------------------------------------------------------

;-------------------------------------------------------------------------------------------------------------------------------
;3.2    Keyboard matrix
;
;The buttons of the keyboard are organized in a matrix. The columns are driven by an i4003 shifter, and the status of selected rows
;can be fetched from the ROM1 input port. There is an 8 state digit point switch and a 3 state rounding switch which are mapped to
;the keyboard matrix through diodes, connecting the 9th and 10th bit of the shifter into the 4 bits of the ROM1 port. This way one
;shortcut on the 8 state digit point switch may activate 0, 1 or 2 inputs on the input port of ROM1, and the switched position is
;directly binary decoded (value 7 can not be set). The 3 state rounding switch can activate 0 or 1 line on the ROM1 port.
;
;The following table summarizes, how the buttons, switches are logically mapped into the 10 columns x 4 rows matrix. This is in
;fact a "mirrored" version of the physical arrangement, as the last column comes first. Behind the name of the buttons there are
;hexadecimal values in parenthesis, which are the assigned scan codes. Only the first 8 columns are scanned for the buttons. The
;state of 9th and 10th column is fetched and just stored into RAM0 as status characters for later processing.
```

```
;
;shifter        ROM1 bit0       ROM1 bit1       ROM1 bit2       ROM1 bit3
;-------------------------------------------------------------------------------------------------------------------------------
;bit0           CM (81)         RM (82)         M- (83)         M+ (84)
;bit1           SQRT (85)       % (86)          M=- (87)        M=+ (88)
;bit2           diamond (89)    / (8a)          * (8b)          = (8c)
;bit3           - (8d)          + (8e)          diamond2 (8f)   000 (90)
;bit4           9 (91)          6 (92)          3 (93)          . (94)
;bit5           8 (95)          5 (96)          2 (97)          00 (98)
;bit6           7 (99)          4 (9a)          1 (9b)          0 (9c)
;bit7           Sign (9d)       EX (9e)         CE (9f)         C (a0)
;bit8           dp0             dp1             dp2             dp3     (digit point switch,
value 0,1,2,3,4,5,6,8 can be switched)
;bit9           sw1             (unused)        (unused)        sw2     (rounding switch, value
0,1,8 can be switched)
;-------------------------------------------------------------------------------------------------------------------------------


;-------------------------------------------------------------------------------------------------------------------------------
;3.3    Printer
;
;The printer contains a continuously rotating cylinder shaped printer drum equipped with 18
columns having 13 characters on each,
;except column 16, which is empty. The first 15 columns form the numbers, the last two columns
give the special characters in the
;following order:
;
;     sector       column 1-15     column 17         column 18
;       0               0               diamond         #
;       1               1               +               *
;       2               2               -               I
;       3               3               X               II
;       4               4               /               III
;       5               5               M+              M+
;       6               6               M-              M-
;       7               7               ^               T
;       8               8               =               K
;       9               9               SQRT            E
;       10              .               %               Ex
;       11              .               C               C
;       12              -               R               M
;
;Rotating of the drum is followed through two input signals. The sector signal becomes active
for each row of characters (sensed
;at the TEST pin of i4004), while the index signal informs the controlling unit, when the first
row is in the printing position
;(led to bit 0 of ROM2 input port). In reality the period of sector signals is around 28ms
(35.7Hz), and the index signal is
;13*28=364ms (2.74Hz). Each column has a separate hammer, which can be fired towards the drum,
when the desired character just
;passes on the corresponding column. The shape of character is printed through an inked ribbon
to the paper. The upper and lower
;half of the ribbon is inked into different colors (black and red), rising of the ribbon is
controlled by bit0 of RAM0 port. The
;hammers are selected through two cascaded i4003 shifter registers, which have the following
connection towards the columns:
;
;bit00          column 17       special characters
;bit01          column 18       special characters
;bit02          -               not used
;bit03          column 1        digit or digit point
;bit04          column 2        digit or digit point
;bit05          column 3        digit or digit point
;bit06          column 4        digit or digit point
```

```
;bit07          column 5        digit or digit point
;bit08          column 6        digit or digit point
;bit09          column 7        digit or digit point
;bit10          column 8        digit or digit point
;bit11          column 9        digit or digit point
;bit12          column 10       digit or digit point
;bit13          column 11       digit or digit point
;bit14          column 12       digit or digit point
;bit15          column 13       digit or digit point
;bit16          column 14       digit or digit point
;bit17          column 15       digit or digit point
;bit18          -               not used
;bit19          -               not used
;
;When the shifter is filled with the correct pattern, and the desired row is in the right position, the hammers can be fired by
;bit1 of RAM0 port. E.g. when SQRT(2)=1.4142135623730 is printed, the following list of data should be sent to the shifter:
;
;       Sector  Hex     binary (msb-lsb)        meaning (1.4142135623730 SQ)
;       ------------------------------------------------------------------------------------------------------------------------
;       0       20000   00100000000000000000                           0        ;one digit 0 is used
;       1       00248   00000000001001001000             1  1  1                ;three digit 1 are used
;       2       02100   00000010000100000000                  2    2            ;two digit 2 are used
;       3       14400   00010100010000000000                    3   3 3         ;three digit 3 are used
;       4       000A0   00000000000010100000               4 4                  ;two digit 4 are used
;       5       00800   00000000100000000000                     5              ;one digit 5 is used
;       6       01000   00000001000000000000                      6             ;one digit 6 is used
;       7       08000   00001000000000000000                         7          ;one digit 7 is used
;       8       00000   00000000000000000000                                    ;digit 8 is not used
;       9       00001   00000000000000000001                             SQ     ;digit 9 is not used, square root character is used
;       10      00010   00000000000000010000              .                     ;digit point
;       11      00000   00000000000000000000                                    ;no character is used from this row
;       12      00000   00000000000000000000                                    ;no character is used from this row
;
;The print operation is started with whatever print row happens to be under the hammers at the time the print function is called.
;The data list shown in the table above may be sent exactly as in that order, or it may be started e.g. with sector 4 and continue
;through 12, 0, and back to 3. If this had not been designed, there would have been a pause of up to a full rotation of the drum
;(in worst case approximately 364ms) before printing would start.
;
;When all columns are printed, the paper can be advanced, which is activated by bit3 of RAM0 port.
;--------------------------------------------------------------------------------------------------------------------------------


;--------------------------------------------------------------------------------------------------------------------------------
;;Chapter 4 SOFTWARE ENVIRONMENT
;--------------------------------------------------------------------------------------------------------------------------------
```

```
;---------------------------------------------------------------------------------------------------
----------------------------------
;4.1    i4004 instruction set summary
;
;Opcode   2nd byte              Mnemonic CY     description
;---------------------------------------------------------------------------------------------------
----------------------------------
;00000000                NOP     -      No operation
;0001CCCC AAAAAAAA       JCN     -      Jump conditional
;0010RRR0 DDDDDDDD       FIM     -      Fetch indirect from ROM into register pair
;0010RRR1                SRC     -      Send Register Control
;0011RRR0                FIN     -      Fetch indirect from ROM (register pair = indirect from
location R0R1 of the same page)
;0011RRR1                JIN     -      Jump indirect (8 bit of program counter = register pair)
;0100AAAA AAAAAAAA       JUN     -      Jump unconditional
;0101AAAA AAAAAAAA       JMS     -      Jump to subroutine
;0110RRRR                INC     -      Increment register
;0111RRRR AAAAAAAA       ISZ     -      Increment register, and jump at nonzero result
;1000RRRR                ADD     CY     Add register and carry to accumulator (ACC=ACC+reg+CY)
;1001RRRR                SUB     CY     Subtract register and borrow from accumulator (ACC=ACC+
(15-reg)+(1-CY))
;1010RRRR                LD      -      Load register into accumulator
;1011RRRR                XCH     -      Exchange register with accumulator
;1100DDDD                BBL     -      Branch back (return) and load data into accumulator
;1101DDDD                LDM     -      Load data into accumulator
;
;11100000                WRM     -      Write accumulator into main memory
;11100001                WMP     -      Output accumulator to RAM port
;11100010                WRR     -      Output accumulator to ROM port
;11100011                WPM     -      Write accumulator to 4008/4009 read/write program memory
(not used in this application)
;11100100                WR0     -      Write accumulator into status character 0
;11100101                WR1     -      Write accumulator into status character 1
;11100110                WR2     -      Write accumulator into status character 2
;11100111                WR3     -      Write accumulator into status character 3
;11101000                SBM     CY     Subtract main memory and borrow from accumulator
(ACC=ACC+(15-mem)+(1-CY))
;11101001                RDM     -      Read main memory into accumulator
;11101010                RDR     -      Input ROM port into accumulator
;11101011                ADM     CY     Add main memory and carry to accumulator
(ACC=ACC+mem+CY)
;11101100                RD0     -      Read accumulator from status character 0
;11101101                RD1     -      Read accumulator from status character 1
;11101110                RD2     -      Read accumulator from status character 2
;11101111                RD3     -      Read accumulator from status character 3
;
;11110000                CLB     0      Clear both (accumulator and carry)
;11110001                CLC     0      Clear carry
;11110010                IAC     CY     Increment accumulator
;11110011                CMC     CY     Complement carry (CY=1-CY)
;11110100                CMA     -      Complement accumulator (ACC=15-ACC)
;11110101                RAL     CY     Rotate accumulator left through carry
;11110110                RAR     CY     Rotate accumulator right through carry
;11110111                TCC     0      Transmit carry to accumulator and clear carry (ACC=CY)
;11111000                DAC     CY     Decrement accumulator
;11111001                TCS     0      Transmit carry subtract and clear carry (ACC=9+CY)
;11111010                STC     1      Set carry
;11111011                DAA     CY     Decimal adjust accumulator (ACC=ACC+6, if CY=1 or ACC>9)
;11111100                KBP     -      Keyboard process (0->0, 1->1, 2->2, 4->3, 8->4, rest-
>15)
;11111101                DCL     -      Designate command line (not used in this application)
;---------------------------------------------------------------------------------------------------
----------------------------------
;
;Meaning of CCCC bits in the JCN instructions:
```

```
;
;       CCCC    hex     abbreviation    jump, when
;       -------------------------------------------
;       0001    1       TZ              test=0
;       0010    2       C1              cy=1
;       0100    4       AZ              accumulator=0
;       1001    9       TN              test=1
;       1010    a       C0              cy=0
;       1100    c       AN              accumulator!=0
;
;Combination of last 3 bits of CCCC would result logic "or" function with the individual conditions, however these are not used in
;this application.
;------------------------------------------------------------------------------------------------------------------------------------
;
;The instruction set does not contain the very basic logical functions ("OR" and "AND"), which may be necessary in the application.
;This function can be implemented in a bit level through using the carry bit and the accumulator functions. It is advantageous, if
;the questioned bit(s) are placed to the lowest or highest bit position(s) (bit 0 and bit 3 in the 4 bit wide registers).
;
;E.g. if an "AND 1" would be needed for testing bit 0, the bit can be tested from the carry after the "RAR" instruction.
;"OR 1" can be replaced with the sequence of "RAR", "STC" and "RAL" instructions.
;
;------------------------------------------------------------------------------------------------------------------------------------

;------------------------------------------------------------------------------------------------------------------------------------
;4.2    Using of ROM areas.
;
;There are 5 i4001 ROMs which implement the program address range of $000-$4ff (divided into 5 pages). The 5th ROM at address $400
;is optional, and contains the implementation of the SQRT function. This ROM is not included, if the end user have not paid for the
;square root key. Even though the relevant "JUN $400" instruction is still in the remaining ROMs, however it would be never called
;due to the missing SQRT physical button.
;
;The memory ranges store the different code parts, which are summarized in the following table. Detailed description of the blocks
;can be found at the referenced memory address.
;
;$000-$027:     main loop
;$029-$04a:     processing of a pressed button
;$04b-$05f:     basic pseudo code engine
;$061-$062:     piece of code for the keyboard matrix handling
;$063-$069:     keyboard shifter handling
;$06a-$076:     printer drum synchronization
;$077-$080:     piece of code for the keyboard matrix handling
;$081-$0a0:     table for translating the keyboard scan codes into function code and parameter
;$0a1-$0af:     table for translating the function code into pseudo code entry address
;$0b0-$0ff:     main part of keyboard matrix handling
;$100-$1f8:     implementation of pseudo instructions
;$1f9-$293:     implementation of pseudo instructions for printer handling
;$294-$2ff:     implementation of pseudo instructions
;$300-$304:     basic pseudo code fetch routines
;$305-$3ff:     pseudo code implementation of the calculator
;$400-$418:     SQRT pseudo code engine
;$419-$427:     unused memory area (NOPs)
;$428-$446:     pseudo code implementation of the SQRT function
;$447-$44f:     unused memory area (NOPs)
;$450-$450:     SQRT pseudo code jump
```

```
;$451-$4aa:     implementation of SQRT pseudo instructions
;$4ab-$4ff:     unused memory area (NOPs)
;-----------------------------------------------------------------------------------------------------------------------------

;-----------------------------------------------------------------------------------------------------------------------------
;4.3    Using of RAM areas:
;
;There are two i4002 RAMs are used, both have 4 registers with 16 cells main memory and 4 cells status characters. In this
;document the altogether eight RAM registers originally designed for storing numbers are generally referred to NR(0) to NR(7),
;(NR(0)..NR(3) in RAM0 and NR(4)..NR(7) in RAM1), the main memory cells are referred to M0..M15 and status characters as S0..S3.
;E.g. NR(7).S2 means status character 2 of register 3 in RAM1.
;
;Generally these registers are places for storing numbers. Usually status character 0 is the plus / minus sign (value 0
;means positive, 1 or 15 means negative number), status character 1 is the place of digit point, and the main memory cells contain
;the number in BCD form (one cell is for one digit). E.g. NR(1) register containing -75.43 is represented with NR(1).S0=15
;(negative number), NR(1).S1=2 (number has 2 digits behind the digit point), and NR(1).M4..M15=0, NR(1).M3=7, NR(1).M2=5,
;NR(1).M1=4 and NR(1).M0=3 (digits are adjusted to main memory location 0 starting with lowest digit value). In this calculator
;implementation the numbers are handled at 14 digit length, therefore M14 and M15 are 0, however temporarily it may contain valid
;data, e.g. during multiplication the two 14 digit numbers can produce 28 digit result. Status character 2 and 3 of the registers
;are used for special purposes.
;
;The number registers has an additional abbreviations as those are typically used for predefined purposes in the implementation of
;this calculator:
;
;NR(0):   KR    keyboard register       keyboard buffer
;NR(1):   WR    working register        input register (usually 2nd operand in add/sub/mul/div operation)
;NR(2):   DR    dividend register       multiplicand/dividend (1st operand of mul/div operation)
;NR(3):   RR    result register         temporary register (result of mul/div/sqrt operation, copied finally into WR)
;NR(4):   CR    constant register       constant multiplicand/dividend register
;NR(5):   SR    sub total register      sub total accumulator (1st operand of add/sub function)
;NR(6):   TR    main total register     main total accumulator (1st operand of add/sub function)
;NR(7):   MR    memory register         memory register (1st operand of memory add/sub function)
;
;Register 0 in RAM0 (KR) is not for storing numbers instead it is used for the keyboard buffer. It may happen during a time
;consuming operation that the user already presses new button(s) on the keyboard. These are stored temporarily into this buffer
;waiting for later processing.
;
;Special meaning of the RAM status characters:
;
;NR(1..7).S0: sign (bit0=0: positive, bit0=1: negative)
;NR(1..7).S1: place of digit point
;
;KR.S0: the keyboard buffer pointer
;KR.S3: keyboard pressing status (0=no button is held down, 15=a button is held down)
;
;WR.S2: rounding switch (10th column of the keyboard matrix) (0=floating, 1=rounding, 8=truncating)
;WR.S3: digit point switch (9th column of the keyboard matrix), values 0,1,2,3,4,5,6 and 8 are used
```

```
;
;DR.S2: multiply/divide status
;RR.S2: last operation
;CR.S2: digit entry mode status
;SR.S2: overflow status
;
;TR.S2: regularly cleared, but never read
;
;unused status characters (remains 0 after reset): KR.S1, KR.S2, DR.S3, RR.S3, CR.S3, SR.S3, TR.S3, MR.S2, MR.S3
;-----------------------------------------------------------------------------------------------------------------------------------------

;-----------------------------------------------------------------------------------------------------------------------------------------
;4.4 Basics of operation and implementation of the calculator
;
;The software of the calculator is written in i4004 assembly code and it uses the "interpretive mode" concept, where another
;instruction set is simulated (called pseudo instruction codes) by fetching the simulated codes from the memory and executing the
;associated routines, which implement the required subfunctions.
;
;In this calculator implementation the "hardware device drivers" for the keyboard and printer are implemented in native
;assembly code, but the main part of calculator logic is coded as a sequence of a pseudo codes in the following way:
;
;The calculator regularly scans the keyboard matrix in the main loop. When any of the buttons is pressed, a function code (in R5),
;a parameter (in R4) and a pseudo code entry address (in R0R1) are assigned to it, and the pseudo code interpreter is called (at
;overflow situation only buttons "CE" and "C" start the engine). In the pseudo code a "state machine" is implemented, where the
;calculator is modeled with internal states and registers. When the pseudo code engine starts to execute the pseudo instructions
;from the defined address, the functions modify these internal states and the number registers heavily depending on the function
;code, parameter and earlier states of state machine. At the end of this operation the pseudo code engine is terminated, and the
;keyboard scanning is continued in the main loop.
;
;In the implemented pseudo code interpreter there is no "routine or function" call possibility, but there is a need to execute the
;same subfunctions from different places. This problem is solved in a way, that the common code is started, and at the end of the
;execution the pseudo code branches to the relevant places, typically based on the function code or parameter. These are more or
;less constant values during the pseudo code execution, however their values may be ruined at the very last checks or as a loop
;counting. Additionally the function code at processing the "M=+" and "M=-" buttons are changing from 8 to 3, which start to work
;as an "=" function and later these are switched to "M+" and "M-" function.
;
;The main states of the state machine are stored in RR.S2, DR.S2, CR.S2 and SR.S2. These have only limited values, which represent
;the following situations:
;
;RR.S2=0 - new number is entered (or some operation is ended, and the result can be used as a new number)
;RR.S2=1 - last operation was multiply or divide
;RR.S2=8 - last operation was addition or subtraction
;
;DR.S2=0 - no started multiplication or division (may mean DR is divided by default)
;DR.S2=3 - started multiplication (DR is multiplied)
;DR.S2=4 - started division (DR is divided)
```

```
;DR.S2=8 - default division with constant value in CR
;DR.S2=B - started multiplication with constant value in CR
;DR.S2=C - started division with constant value in CR
;
;CR.S2=0 - digit entry is not started yet
;CR.S2=1 - started digit entry with digit point
;CR.S2=8 - started digit entry without digit point
;
;SR.S2=0 - no overflow
;SR.S2=1 - overflow occurred
;
;Below is a summarization, which inputs are assigned to the pressed buttons and how the
registers and the main states are modified
;based on their previous values. Following the "instructions" defined by this table the
operation of the calculators can be
;understood. It does not contain the printing and digit point adjustment instructions, just the
high level handling of the main
;states and the number registers. "-" in the in columns means not checked state, while "-" in
the out columns means not changed
;value. Short remark is also added ("M=+" and "M=-" are not commented, there the same comments
are valid like at "=" and at the
;memory add/sub function).
;
;------------------------------------------------------------------------------------------------
-----------------------------------
;       entry   in      in      in      in      out     out
;button address fcode   fpar    RR.S2   DR.S2   RR.S2   DR.S2   operation
(remark)
;------------------------------------------------------------------------------------------------
-----------------------------------
;*      $306    1       3       0       0       1       3       DR=WR, CR=WR
(save the first operand)
;                               0       3       1       3       RR=DR*WR, WR=RR, DR=WR, CR=WR
(chain mul/div operation)
;                               0       4       1       3       RR=DR/WR, WR=RR, DR=WR, CR=WR
(chain mul/div operation)
;                               0       8,B,C   1       3       DR=WR, CR=WR
(new number after const mul/div)
;                               1       -       1       3       DR=WR, CR=WR
(operation correction)
;                               8       -       1       3       WR=TR, DR=WR, CR=WR, SR=0, TR=0
(main total is the first operand)
;
(clear totals)
;/      $306    1       4       0       0       1       4       DR=WR, CR=WR
(save the first operand)
;                               0       3       1       4       RR=DR*WR, WR=RR, DR=WR, CR=WR
(chain mul/div operation)
;                               0       4       1       4       RR=DR/WR, WR=RR, DR=WR, CR=WR
(chain mul/div operation)
;                               0       8,B,C   1       4       DR=WR, CR=WR
(new number after const mul/div)
;                               1       -       1       4       DR=WR, CR=WR
(operation correction)
;                               8       -       1       4       WR=TR, DR=WR, CR=WR, SR=0, TR=0
(main total is the first operand)
;
(clear totals)
;+      $391    2       1       0,8     -       8       -       RR=WR, TR=TR+WR, SR=SR+WR
(add number to totals)
;                               1       -       8       0       RR=WR, TR=TR+WR, SR=SR+WR
(add number to totals)
;
;-      $391    2       2       0,8     -       8       -       RR=WR, TR=TR-WR, SR=SR-WR
(sub number from totals)
;                               1       -       8       0       RR=WR, TR=TR-WR, SR=SR-WR
```

```
(sub number from totals)
;
;M+     $398    3       5       -       -       0       -       RR=WR, MR=MR+WR
(add number to memory)
;
;M-     $398    3       6       -       -       0       -       RR=WR, MR=MR-WR
(sub number from memory)
;
;EX     $3f1    4       a       -       -       0       -       CR=WR, RR=DR, DR=WR, WR=RR
(exchange)
;
;diamond $3cd   5       0       0       -       -       -       (print only)
;                               1       -       -       -       (print only)
;                               8       -       -       -       WR=SR, SR=0
(recall subtotal)
;
;00     $3d7    6       0       -       -       0       -       WR=add two new zeros to the
number  (number entry)
;
;RM     $3fd    7       c       -       -       0       -       WR=MR
(recall memory)
;
;=      $38a    8       1       0,1     0,4     0       +8      CR=WR, RR=DR/WR, DR=?, WR=RR
(divide, divisor to const)
;                               0,1     3       0       +8      RR=DR*WR, DR=0, WR=RR
(multiply)
;                               0,1     8,C     0       -       DR=WR, WR=CR, RR=DR/WR, WR=RR
(const divide)
;                               0,1     B       0       -       DR=WR, WR=CR, RR=DR*WR, WR=RR
(const multiply)
;                               8       -       0       -       WR=TR, SR=0, TR=0 (recall main
total, clear totals)
;
;M=+    $38a    8->3    5       0,1     0,4     0       +8      CR=WR, RR=DR/WR, DR=?, WR=RR,
RR=WR, MR=MR+WR
;                               0,1     3       0       +8      RR=DR*WR, DR=0, WR=RR,
RR=WR, MR=MR+WR
;                               0,1     8,C     0       -       DR=WR, WR=CR, RR=DR/WR, DR=?,
WR=RR,  RR=WR, MR=MR+WR
;                               0,1     B       0       -       DR=WR, WR=CR, RR=DR*WR, DR=0,
WR=RR,  RR=WR, MR=MR+WR
;                               8       -       0       -       WR=TR, SR=0, TR=0,
RR=WR, MR=MR+WR
;
;M=-    $38a    8->3    6       0,1     0,4     0       +8      CR=WR, RR=DR/WR, DR=?, WR=RR,
RR=WR, MR=MR-WR
;                               0,1     3       0       +8      RR=DR*WR, DR=0, WR=RR,
RR=WR, MR=MR-WR
;                               0,1     8,C     0       -       DR=WR, WR=CR, RR=DR/WR, DR=?,
WR=RR,  RR=WR, MR=MR-WR
;                               0,1     B       0       -       DR=WR, WR=CR, RR=DR*WR, DR=0,
WR=RR,  RR=WR, MR=MR-WR
;                               8       -       0       -       WR=TR, SR=0, TR=0,
RR=WR, MR=MR-WR
;
;SQRT   $305    9       1       -       -       0       0       CR=WR, RR=SQRT(WR), DR=?, WR=RR
SQRT
;
;%      $361    a       1       -       0,4     0       +8      CR=WR, RR=DR/WR, DR=?, WR=RR
(divide, divisor to const)
;                               -       3       0       +8      RR=DR*WR, DR=0, WR=RR
(multiply)
;                               -       8,C     0       -       DR=WR, WR=CR, RR=DR/WR, DR=?,
WR=RR   (const divide)
;                               -       B       0       -       DR=WR, WR=CR, RR=DR*WR, DR=0,
WR=RR   (const multiply)
```

```
;
;CM     $3f9    b       b       -       -       0       -       WR=MR, MR=0
(recall memory and clear)
;
;000    $3d7    c       0       -       -       0       -       WR=add three new zeros to the
number  (number entry)
;
;digit  $3d7    d       digit   -       -       0       -       WR=add new digit to the number
(number entry)
;
;sign   $3d7    d       10      -       -       0       -       WR=change the sign of WR
(number entry)
;
;dp     $3d7    d       11      -       -       0       -       WR=mark the digit point
(number entry)
;
;CE     $3ca    e       0       -       -       0       -       WR=0
(entry clear)
;
;C      $3c5    f       b       -       -       0       0       WR=0, DR=0, SR=0, TR=0
(clear operands and totals)
;-------------------------------------------------------------------------------------------------
-----------------------------------
;
;As an example let us follow how the "24 * 3 =" is executed. Assumed, that at the beginning all
the internal states and registers
;are cleared.
;
;-------------------------------------------------------------------------------------------------
-----------------------------------
;       in      in      in      in      out     out
;button fcode   fpar    RR.S2   DR.S2   RR.S2   DR.S2   operation       remark
;-------------------------------------------------------------------------------------------------
-----------------------------------
;2      d       2       0       0       0       0       WR=2            digit is placed to the
working register
;4      d       4       0       0       0       0       WR=24           digit is added to the
end of the working register
;*      2       1       0       0       1       3       DR=24           number is copied to
dividend/multiplicand register
;                                                       CR=24           number is copied to
constant register
;3      d       3       1       3       0       3       WR=3            digit is placed to the
working register
;=      8       1       0       3       0       B       RR=24*3=72      multiplication is
calculated
;                                                       DR=0            multiplicand is cleared
;                                                       WR=72           result is copied to the
working register
;-------------------------------------------------------------------------------------------------
-----------------------------------
;Note: CR still contains the original multiplicand, new numbers can be multiplied with it.
DR.S2=B indicates this.
;
;Note: The calculator can be reset by pressing the "CM", "C" and "Ex" buttons in this order.
"CM" is required to clear MR, "C"
;clears WR, DR, SR and TR, and "Ex" copies the cleared WR, DR into RR and CR. These clear the
internal flags too. Pressing just "C"
;may be also enough (if clearing the memory is not needed), but it leaves numbers in RR and CR,
which are not disturbing, as those
;can not be referred in any way (those will be overwritten with new values before use).
;-------------------------------------------------------------------------------------------------
-----------------------------------

;-------------------------------------------------------------------------------------------------
-----------------------------------
```

```
;4.5 Square root implementation
;
;Mathematical background:
;
;At first, let us think in the set of integer numbers only. The task is to find a suitable "p"
to certain "N", where
;
;       (p+1)*(p+1) > N >= p*p.  We can say then, that SQRT(N)=p.
;
;E.g. SQRT(5936123) would lead to 2436, as 2437*2437=5938969 > 5936123 >= 2436*2436=5934096.
;
;It is easy to see, that stealing 2 digits from the end of "N" results stealing one digit from
the end of SQRT(N):
;
;       E.g. as SQRT(5936123)=2436, then SQRT(59361)=243, SQRT(593)=24 and SQRT(5)=2.
;
;The square root is guessed digit by digit from left to right in the reverse logic to the
previous rule. For calculating the sqrt
;for a big number, at first digits are "removed" pair by pair from the end, e.g. 5936123 leads
to 5, then the calculation is
;started from the "simplified" number (from 5). When the square root is already known for it
("half result"), then the next
;two digits of "N" are taken (93) and the next digit of the square root of 593 is "guessed"
based on the already known "half
;result". This algorithm is repeated till the end of the "N" by taking the removed digit pairs
(61 and 23). (Bigger accuracy can
;be reached, if "N" is extended with "0"-s, and the same algorithm is continued.)
;
;For demonstrating how the next digit is "guessed", let us assume, that the square root is
already calculated to certain number
;of digits. E.g. we already calculated SQRT(593)=24, and we would like to find the next digit
("d") in SQRT(59361), which must
;be in the form of 240 + d, where d=0..9.
;
;The base of square root algorithm is implemented by using the following equation:
;
;       N >= (a+b)*(a+b) = a*a + 2*a*b + b*b, which is in other form:
;
;       (N - a*a) - (2*a*b + b*b) >= 0.
;
;In our example "N"=59361, "a" is 240, and "b" is the next digit "d", or with the actual
numbers:
;
;       (59361 - 240*240) - (2*240*d + d * d) >= 0, which is in shorted way
;
;       1761 - (480*d + d*d) >= 0.
;
;In this equation "d" is tried from 1 to 10, and the loop is stopped, when the left side becomes
negative. If we are doing it
;one by one, we can save the multiplication in the "480*d", if we subtract 480 at every step
from 1761. Calculation of d*d
;can be simplified too. Let us notice, that the difference between the neighboring square
numbers are an odd number,
;d*d-(d-1)*(d-1)=d*d-(d*d-2*d+1)=2*d-1, thus by adding the odd numbers from 1,3,5 ... serially
we get the series of square
;numbers (1=1*1, 1+3=2*2, 1+3+5=3*3, ... 1+3+5+7+9+11+13+15+17+19=10*10). If the odd numbers are
serially added to 480, the
;result of subtraction will contain also sum of the odd numbers, that is the d*d square.
Alternatively after starting the
;subtrahend value with 480+1, it can be incremented by 2 at every step.
;
;In our example the next digit would be determined in the following way:
;
;step   remainder       try     subtrahend      altogether with previous subtrahends        new
remainder
;1          1761        d=1     480+1           480             + 1         = 480*1 + 1*1     1280
```

```
;not negative
;2          1280      d=2     480+3           480+480         + 1+3     = 480*2 + 2*2      797
;still not negative
;3           797      d=3     480+5           480+480+480     + 1+3+5   = 480*3 + 3*3      312
;still not negative
;4           312      d=4     480+7           480+480+480+480 + 1+3+5+7 = 480*4 + 4*4     -175
;negative, stop
;
;This means, that the next digit should be 3, as 4 already produces negative remainder, but 3
does not, so SQRT(59361)=243.
;Note, that the "new remainder" (312) gives the hundreds of the initial remainder for the next
round (59361-243*243=312
;or after taking the next two digits 5936123-2430*2430=31200+23). See also, that the subtrahend
at step 4 is bigger with 1,
;than the tens of subtrahend for the next round (2*243*10=4860), an additional decrement is
needed before the next round.
;
;For starting the algorithm the first one or two digits of the number are taken as the initial
remainder, the initial half result
;is cleared. In our example, this would look like with the following numbers (round 5 and 6 is
not necessarily needed, it just
;demonstrates, how further accuracy can be reached):
;
;round   number    digit pair  remainder   half result   guessed digit   subtracted numbers
new remainder
;1       5              5             5           0             2       1,3
1
;2       593          93           193           2             4       41,43,45,47
17
;3       59361        61          1761          24             3       481,483,485
312
;4       5936123      23         31223         243             6
4861,4863,4865,4867,4869,4871            2027
;5       593612300    00        202700        2436             4       48721,48723,48725,48727
7804
;6       59361230000  00        780400       24364             1       487281
293119
;
;Thus SQRT(5936123)=2436 (or 2436.41, if two more digits are needed). If we are thinking in real
numbers instead of integers,
;similar algorithm can be used. Now the digit pairing at the first simplification should be
started from the digit point to left
;and right direction too. E.g. in "593.6123" the digit pairing would look like this "5  93  .
61  23". Then the
;algorithm can be started similarly, just when the digit point is reached between the digit
pairs, a digit point has to be placed
;into the result too, but apart from this, the calculation should be done as integer numbers
would be handled. When SQRT(593)=24
;is determined, the digit point is added right after the interim result "24.", and the next
guessed digits are placed behind it,
;that is sqrt(593.6123)=24.3641.
;
;In the real implementation square root of WR is calculated. WR is left shifted into the
leftmost position, then it is copied to
;DR. Due to the digit pairing the shift may be needed one more time again, which is done in DR
in this case. The new place of
;digit point is calculated based on the original place of digit point and the number of shifts,
which is counted in R10 and R11.
;WR is the place of subtrahend, which is cleared for starting the algorithm. Multiplication with
10 is implemented with shifting
;the number (as BCD arithmetic is used) or it is not needed at all, as R15 dynamically points to
the part of the number, which is
;already used in the calculation. DR contains the remainder. The "guessed" digit is counted in
R13 and is shifted into RR. The
;final result is copied from RR to WR after returning back to the basic pseudo code interpreter
to address $340. Place of the digit
```

```
;point is set from R11 there too.
;
;Note: this implementation does not care about the sign, SQRT(X) is handled generally by
SQRT(ABS(X)).
;-----------------------------------------------------------------------------------------------
-----------------------------------


;-----------------------------------------------------------------------------------------------
-----------------------------------
;4.6 Unusual results
;
;Strange behaviors: The following list contains few sequences, which produces unusal results.
Some of these can be planned
;or this kind of malfunctions are intentionally left in the program due to the memory size
limitation (correct handling would have
;required more code), or maybe the target of this analysis was a slightly damaged assembly code
(bits of the original ROMs may be
;damaged during the long years). Below there are some cases, which can be reproduced from the
just powered up calculator (assumed,
;that the calculator is switched to floating and 0 digit point state).
;
;1, The default operation is the division. If "10 Ex 3 =" is entered, the result is
"3.3333333333333", which is 10/3.
;
;2, If "=" is entered, then an overflow occurs due to the default division operation and the 0/0
calculation.
;
;3, DR internal register is not initialized correctly at the end of a division, when "=", "M=+",
"M=-" or "%" is pressed. This
;   time DR contains the left adjusted remainder, which may use the 15th digit position too,
while the place of digit point is
;   inherited from the previous dividend. This number can be exchanged into WR, and can be the
source for other operations.
;   Entering "10 / 3 = Ex Diamond" results "100000000000000" (15 digit length number!). "10.01 /
3 = Ex Diamond" places
;   "2000000000000.00" into WR which is printed as "000000000000.00", as there is no column for
the leading "2" (digit point
;   already uses an extra column), but "10.01 / 3 = Ex / 3 =" gives "666666666666.6".
;
;4, Square root function leaves in DR the double of root result with cleared digit point. This
also can be 15 digit number, it can
;   be exchanged into WR, or can be the source of the default division. E.g. "2 SQRT Ex Diamond"
gives "28284271247460", or
;   "2 SQRT 3 =" and repeatedly pressing new "="-s produces series of very strange numbers.
(This might be due to one bit damage at
;   address $4aa, in this code there is $04, but originally $0c is very probable. If the
assumption is correct, DR would get the
;   original value of WR.)
;
;5. Digit point adjustment problem. Adding and subtracting does not check the place of digit
point, it assumes, that both operands
;   are adjusted to the same digit point place (determined by the digit point switch). However
changing the state of a digit point
;   during an operation will cause incorrectly added or subtracted numbers, as the changed state
of digit point switch modifies
;   the later entered WR, but the earlier accumulated results in SR, TR or MR are not adjusted,
just the digit point place will be
;   refreshed by the next operation. E.g. "(DP=0) 1 + (DP=3) 2 + =" gives "2.001", where "
(DP=n)" means digit point switch is at
;   position "n".
;-----------------------------------------------------------------------------------------------
-----------------------------------


;-----------------------------------------------------------------------------------------------
-----------------------------------
;4.7 Basic pseudo instruction codes
```

```
;
;The following table summarizes the instructions implemented by the pseudo codes. The first
column is a count, how many times the
;instruction is used from page $300, 2nd column is the hexadecimal value of the pseudo
instruction code, 3rd is an 'artificial'
;mnemonic (or a list of mnemonics), and finally a short description is about the function of the
instruction. In the instructions
;new abbreviations are introduced for some i4004 registers, status characters and their values.
;
;WR.S2: 0=NTRUNC, bit0=ROUND
;DR.S2=MOP: 0=MOPN, bit0=MOPMUL, bit3=MOPCONST
;RR.S2=MODE: 0=MODENN, bit0=MODEMD, bit3=MODEAS
;CR.S2=MENT: bit0:MENTDP
;SR.S2=OVFL
;R13=DIGIT
;R10R11=DPCNT
;
;-----------------------------------------------------------------------------------------------
-----------------------------------
;1 $01  MOV IR,WR       Move working register into indirect register (IR=WR)
;
;3 $02  MOV CR,WR       Move working register into constant register (CR=WR)
;
;3 $03  MOV RR,WR       Move working register into result register (RR=WR)
;
;3 $04  MOV DR,WR       Move working register into dividend/multiplicand register (DR=WR)
;
;2 $09  MOV WR,MR       Move memory register into working register (WR=MR)
;
;0 $0A  MOV WR,TR       Move main total register into working register (WR=TR)
;
;1 $0B  MOV WR,SR       Move sub total register into working register (WR=SR)
;
;2 $0C  MOV WR,CR       Move constant register into working register (WR=CR)
;
;3 $0D  MOV WR,RR       Move result register into working register (WR=RR)
;
;2 $0E  MOV WR,DR       Move dividend/multiplicand register into working register (WR=DR)
;
;2 $1E  ADD IR,WR       Add working register to indirect register (IR=IR+WR)
;
;2 $21  ADD DR,WR       Add working register to dividend/multiplicand register (DR=DR+WR)
;
;1 $2C  SUB WR,IR + JPC NNEG + INC DIGIT
;                       Subtract indirect from working register (WR=WR-IR)
;                       jump at non negative with increment the digit
;
;2 $31  SUB IR,WR + JPC NNEG + INC DIGIT
;                       Subtract working register from indirect register (IR=IR-WR)
;                       jump at non negative with increment the digit
;
;1 $34  SUB DR,WR + JPC NNEG + INC DIGIT
;                       Subtract working register from dividend/multiplicand register (DR=DR-WR)
;                       jump at non negative with increment the digit
;
;1 $44  CLR MR          Clear memory register (MR=0)
;
;0 $45  CLR TR          Clear main total register (TR=0)
;
;1 $46  CLR SR          Clear sub total register (SR=0)
;
;0 $47  CLR CR          Clear constant dividend/multiplicand register (CR=0)
;
;1 $48  CLR RR          Clear result register (RR=0)
;
;2 $49  CLR DR          Clear dividend/multiplicand register (DR=0)
```

```
;
;2 $4A  CLR WR          Clear working register (WR=0)
;
;3 $51  SHL RR          Left shift of result register with R13
;
;3 $52  SHL DR          Left shift of dividend/multiplicand register with R13
;
;3 $53  SHL WR          Left shift of working register with R13
;
;2 $5A  SSR RR          Right shorted shift of result register
;                       (one digit right shift of 14 digit length RR with R13 (R13 is shifted
into digit 14))
;
;2 $5D  SHR RR          Right shift of result register
;                       (one digit right shift of RR with R13 (0 is shifted from right))
;
;2 $5E  SHR DR          Right shift of dividend/multiplicand register
;                       (one digit right shift of DR with R13 (0 is shifted from right))
;
;3 $5F  SHR WR          Right shift of working register
;                       (one digit right shift of WR with R13 (0 is shifted from right))
;
;4 $6C  JPC MODENN      Jump, if new number is entered and not processed with add/div/mul/div
(jump, if RR.S2=0)
;
;1 $6D  JPC MOPN        Jump, if divide or multiply operation is not specified (jump, if
DR.S2=0)
;
;1 $6E  JPC NTRUNC      Jump, if number is not truncated/rounded (jump, if WR.S2=0)
;
;0 $73  JPC OVFL        Jump at overflow (jump, if SR.S2.bit0<>0)
;
;1 $74  JPC MENTDP      Jump if number is entered with digit point (jump, if CR.S2.bit0<>0)
;
;4 $75  JPC MODEMD      Jump, if number is used for mul/div operation (jump, if RR.S2.bit0<>0)
;
;2 $76  JPC MOPMUL      Jump, if multiplication is started (jump, if DR.S2.bit0<>0)
;
;1 $77  JPC ROUND       Jump, if rounding is needed (jump, if WR.S2.bit0<>0)
;
;2 $7B  JPC MOPCONST    Jump, if multiplication/division is done with constant value (jump, if
DR.S2.bit3>0)
;
;1 $7F  CLR OVFL        Clear overflow (SR.S2=0)
;
;2 $82  CLR MOP         Clear divide/multiply operation (clear DR.S2=0)
;
;1 $85  SET OVFL        Set overflow (SR.S2=1)
;
;1 $86  SET MENTDP      Set that number is entered with digit point (set CR.S2=1)
;
;1 $87  SET MODEMD      Set that number is used for mul/div operation (set RR.S2=1)
;
;1 $8A  SET MODEAS      Set that number is used for add/sub operation (set RR.S2=8)
;
;3 $8D  SET MOPPAR      Set the multiplication/division from function parameter (set
DR.S2=function code parameter)
;
;1 $90  SET MOPCONST    Set that multiply/divide operation is with constant value (set
DR.S2.bit3=1)
;
;1 $97  JPC NBIG_IR     Jump if indirect register does not contain big value (upper two digits
are empty)
;
;4 $9A  JPC NBIG_WR     Jump if working register does not contain big value (upper two digits
are empty)
```

```
;
;1 $9E  CLR DIGIT + JPC NBIG_DR
;                       Clear the digit (R13=0)
;                       Jump if dividend/multiplicand register does not contain big value (upper
two digits are empty)
;
;2 $A0  CLR DIGIT + JPC ZERO_DR
;                       Clear the digit (R13=0)
;                       Jump if dividend/multiplicand register is zero
;
;1 $A2  JPC ZERO_WR     Jump if working register is zero
;
;16 $A7 JMP             Jump always
;
;3 $A9  JPC BIG_DIGIT   Jump, if digit is bigger then 9 (jump, if R13>9)
;
;1 $AC  JPC ZERO_DIGIT + DEC DIGIT (decrement R13 and jump, if R13 was 0 before the decrement)
;
;1 $AE  CLR DIGIT + JMP         Clear digit (R13=0) and jump
;
;2 $B1  JPC NEWOP       Jump at new add/sub/mul/div operation (jump, if function code<8)
;
;3 $B4  JPC MEMOP       Jump at new memory operation (jump, if function parameter>3)
;
;2 $B7  JPC ROTFC       Rotate the function code right and jump, if the rotated out bit is zero
;
;2 $BC  JPC ODDPAR      Jump if function parameter is odd
;
;1 $BF  SET DP_IR       Set digit point place of indirect register (IR.S1=R11)
;
;2 $C2  SET DP_WR       Set digit point place of working register (WR.S1=R11)
;
;1 $C6  GET DP_WR       Get the digit point place of working register (R11=WR.S1)
;
;5 $CA  INC DPCNT       Increment digit point counter (increment R10R11)
;
;3 $CE  JPC NBIG_DPCNT  Jump, if digit point counter does not exceed the upper limit (jump, if
R10R11<0E)
;
;3 $CF  JPC ZERO_DPCNT  Jump, if digit point counter is zero (jump, if R10R11=0)
;
;1 $D4  JPC DIFF_SIGN   Jump, if working register and indirect register have different sign
;                       (jump, if WR and IR have different sign)
;
;1 $D7  DIGIT           Digit functions
;
;3 $D9  MOV WR,TR + CLR TR + CLR SR     WR=TR, TR=0,  SR=0
;
;1 $DB  SET MRMFUNC + JMP       Set function code the memory function (3) and jump
;
;5 $DD  DEC DPCNT       Decrement digit point counter (decrement R10R11)
;
;1 $DF  GET DPDIFF      Difference between required an actual digit point is set into digit
point counter
;                       (WR.S1=WR.S3, R10R11=difference between required an actual digit point)
;
;1 $E1  GET DPCNTDIV    Digit point counter adjust for division (set R10R11 to DR.S1+(13-R11)-
WR.S1)
;
;1 $E3  GET DPCNTMUL    Digit point counter adjust for multiplication
;                       (set R10R11 to the sum of digital places (WR, DR & current in R11)
;
;2 $E5  SET DIVMUL_SIGN + MOV DIGIT,15
;                       Sign of result register is set based on the WR and DR for multiplication
or division
;                       DIGIT is set to 15  (R13=15, used for loop counting)
```

```
;
;2 $E7  NEG WR           Change the sign of working register (complement WR.S0)
;
;1 $E9  ROUNDING         Increment working register if DIGIT>4
;                        (if R13>4 then increment R14 else increment WR)
;
;1 $EB  PRN ADVANCE + CLR DPCNT
;                        Advancing the printer paper.
;                        (end of printing with advancing the paper and R10R11=0, R14R15=0)
;
;1 $ED  SQRT             Square root of working register is calculated into result register.
Continues at address $40.
;
;3 $EF  CLR MENT + CLR OVFL + RET       Return (CR.S2=0, SR.S2=0, TR.S2=0 and exit)
;
;7 $F1  CLR MODE + CLR MENT + RET       Return (RR.S2=0, CR.S2=0 and exit)
;
;4 $F3  CLR MODE + RET                  Return (RR.S2=0, CR.S2=0 and exit)
;
;0 $F9  PRN FPAR,C       print number with function parameter and char=11 "C" in last column (not
used)
;
;2 $FA  PRN FPAR,MEM     print number with function parameter and char=12 "M" in last column
;
;1 $FB  PRN FPAR,FCODE   print number with function parameter and empty character in last column
;
;5 $FC  PRN FPAR         print number with function parameter and empty character in last column
;
;3 $FD  PRN ROUND,FPAR   print number with optional rounding char and function parameter in last
column
;
;2 $FE  PRN FCODE        print number with function code and empty character in last column
;
;1 $FF  PRN OVFL         print unimplemented number (dots with empty extra columns)
;-----------------------------------------------------------------------------------------------
-----------------------------------

;-----------------------------------------------------------------------------------------------
-----------------------------------
;4.8 Square root pseudo instruction codes
;
;The optional square root ROM implements a second pseudo code engine at address space $400-$4FF,
and uses some of the basic pseudo
;instructions just under new code value, plus implements new instructions. These are:
;
;-----------------------------------------------------------------------------------------------
-----------------------------------
;1 $51  PRN FCODE        print number with function code and empty character in last column
;
;1 $53  CLR RR           Clear result register (RR=0)
;
;1 $55  CLR WR           Clear working register (WR=0)
;
;1 $57  SHL RR           Left shift of result register with R13
;
;1 $58  SHL DR           Left shift of dividend/multiplicand register with R13
;
;1 $59  SHL WR           Left shift of working register with R13
;
;2 $5B  MOV DR,WR        Move working register into dividend/multiplicand register (DR=WR)
;
;1 $5D  SUB DR,WR + JPC NNEG + INC DIGIT
;                        Subtract working register from dividend/multiplicand register (DR=DR-WR)
;                        jump at non negative with increment the digit;
;
;1 $5F  ADD DR,WR        Add working register to dividend/multiplicand register (DR=DR+WR)
```

```
;
;1 $61  JPC ZERO_WR      Jump if working register is zero
;
;1 $63  JPC NBIG_WR      Jump if working register does not contain big value (upper two digits
are empty)
;
;1 $65  CLR DIGIT + GET DP_WR   Clear digit (R13=0) and get the digit point place of working
register (R11=WR.S1)
;
;1 $6A  SET LPCSQRT + SET DPCNTSQRT + JPC EVENDP
;                        Set sqrt loop counter (R15=13)
;                        Adjust place of digit point for sqrt (R10R11=(R10R11/2+6+((R10R11 mod
2))))
;                        Jump, if original place of digit point was even
;
;2 $7A  INC WR_POS       Increment working register from position in R15
;
;1 $85  DEC WR_POS       Decrement working register from position in R15
;
;1 $93  INC DPCNT + JMP increment digit point counter (R10R11) and unconditional jump
;
;1 $96  JMP              Unconditional jump
;
;1 $98  JPC NZERO_LPCSQRT + DEC LPCSQRT decrement sqrt loop counter (R15), and jump, except when
R15 was 0
;
;1 $9C  SHR WR           Right shift of working register
;                        (one digit right shift of WR with R13 (0 is shifted from right))
;
;1 $9F  CLR MOP + RET_BPC        Return back to basic pseudo code interpreter to address $40
;
;1 $A7  MOV CR,WR        Move working register into constant register (CR=WR)
;
;1 $A9  MOV DR,WR        Move working register into dividend/multiplicand register (DR=WR) ???
;       (MOV WR,CR)      It is very probable that this would be move constant register into
working register (WR=CR)
;                        (this way there are two codes for "DR=WR" ($5B and $A9) and here "WR=CR"
would be more logical)
;-------------------------------------------------------------------------------------------------
-----------------------------------

;-------------------------------------------------------------------------------------------------
-----------------------------------
;Chapter 5 Detailed analysis of the assembly code
;-------------------------------------------------------------------------------------------------
-----------------------------------

;-------------------------------------------------------------------------------------------------
-----------------------------------
;Main Loop
;
;At power up the PC is set to 000, and also the internal registers and RAM areas are cleared,
thus no real initialization is needed
;here. The program can be directly started with the main loop including the keyboard handling
and printer drum synchronization.
;When a pressed key is processed, the program will continue the execution in the main loop with
the "jump 000" instruction.
;-------------------------------------------------------------------------------------------------
-----------------------------------

000 f0    junb_000: clb
001 11 01 jcnb_001: jcn TZ $001         ;wait for the inactive printer drum sector signal
003 50 b0           jms $0b0            ;Keyboard handling
005 51 5f jcnb_005: jms $15f            ;right shift of keyboard buffer through R13
007 ad              ld 13
008 b1              xch 1               ;R1=lower half of the possible scan code
```

```
009 f0              clb
00a 51 5f           jms $15f           ;right shift of keyboard buffer through R13
00c ad              ld 13              ;ACC=upper half of the possible scan code
00d 1c 29 jcnb_00d: jcn AN $029        ;jump, if valid data was shifted from the buffer

;Status light handling
;RAM1 port:
;       BIT0 = Memory lamp      (MR)
;       BIT1 = Overflow lamp    (SR.S2.bit0)
;       BIT2 = Minus sign lamp  (WR.S0.bit0)

00f 68              inc 8              ;R4R5 points to WR
010 51 73           jms $173           ;read the overflow bit, CY=SR.S2.bit0
012 27    jcnf_012: src 3<
013 ec              rd0                ;read WR.S0 (minus/positive sign, bit 0 is used)
014 f5              ral
015 b3              xch 3              ;R3=WR.S0 << 1 + (overflow bit)
016 68              inc 8              ;R4R5 points TR
017 f0              clb
018 51 a0           jms $1a0           ;check, whether MR contains any number
01a f3              cmc                ;after negate CY=1, if MR is not empty
01b b3              xch 3
01c f5              ral                ;shift into ACC (8*WR.S0.bit1 + 4*WR.S0.bit0 + 2*
(overflow) + (MR ? 1 : 0))
01d e1              wmp                ;output into RAM1 port
01e 66              inc 6
01f 27              src 3<
020 ea              rdr                ;read ROM2 port
021 f5              ral
022 f7    jcnb_022: tcc
023 14 00           jcn AZ $000        ;jump back, if ROM2.bit3 is low: paper advance button is
not held down
025 52 46           jms $246           ;more advancing the printer paper
027 40 00           jun $000           ;jump back to main loop

;
;A pressed button is found
;

029 b0              xch 0              ;R0R1=Keyboard scan code
02a ec              rd0                ;decrement the keyboard buffer pointer (KR.S0) by two
02b f8              dac
02c f8              dac
02d e4              wr0
02e 27              src 3<
02f ea              rdr                ;read the content of ROM1 port (decimal point switch)
030 e7              wr3                ;write it into WR.S3 (number of decimal places)
031 50 64           jms $064           ;shift one high bit into keyboard shifter
033 27              src 3<
034 ea              rdr                ;read the content of ROM1 port (rounding switch)
035 e6              wr2                ;write it into WR.S2
036 34              fin 2<             ;translate the scan code into function code and
parameter (into R4R5)
037 20 a0           fim 0< $a0
039 a5              ld 5
03a b1              xch 1
03b 30              fin 0<             ;fetch the pseudo code entry address of the function
code from table $0a0-$0af (into R0R1)
03c 68              inc 8
03d 51 73           jms $173           ;read the overflow bit, CY=SR.S2.bit0
03f d0              ldm 0
040 e1              wmp                ;put RAM1.port=0 (clear status lamps)
041 d1              ldm 1              ;ACC=1
042 f3              cmc                ;CY=!(overflow)
043 f5              ral                ;ACC=3 (!overflow) or ACC=2 (overflow)
044 fc              kbp                ;ACC=15 (!overflow) or ACC=2 (overflow)
```

```
045 85              add 5                 ;adding the function code
                                          ;if there is no overflow, all functions set the CY flag
                                          ;if there is overflow, only "C" or "CE" functions set
the CY flag
046 1a 00           jcn C0 $000           ;jump, if overflow blocks the new function
048 f0              clb
049 00              nop
04a 00              nop

;-----------------------------------------------------------------------------------------------
-----------------------------------
;Basic pseudo code engine with keyboard handling
;
;       usage of registers:
;               R0R1 - pseudo code instruction pointer
;               R2R3 - pseudo instruction code
;               R4   - parameter (defined by the last pressed button)
;               R5   - function code (defined by the last pressed button)
;               R6R7 - $20 - points to DR
;               R8R9 - $10 - points to WR
;               R12  - printer drum sector counter
;               ACC  - 0
;               CY   - 0
;
;               R10,R11,R13,R14,R15 - generally usable registers
;               (R10R11 - digit point counter)
;               (R13 - digit, used for shifters, loop counting)
;               (R14 - rounding indicator)
;
;Pseudo code interpreter logic:
;
;Pseudo instruction codes are fetched from the address 300-3ff, based on the R0R1 instruction
pointer. The pseudo instruction codes
;are executed as CPU native assembly instructions by calling a subroutine and jumping to address
$100+code. At the end of the
;execution of a pseudo instruction, the pseudo code instruction pointer is incremented by 1. If
the previous pseudo instruction
;returned ACC with 0 value, the execution is continued from the incremented address, otherwise
the data byte on the incremented
;address is understood as a pseudo code jump address, which is conditionally executed. If the
previously returned CY was 1, it is
;copied into the instruction pointer, otherwise it is skipped by increasing the instruction
pointer again.
;-----------------------------------------------------------------------------------------------
-----------------------------------

04b 11 4f jcnb_04b: jcn TZ $04f           ;wait for the inactive printer drum sector signal
04d 50 b0 iszb_04d: jms $0b0              ;keyboard handling
04f 26 20 jcnf_04f: fim 3< $20
051 28 10           fim 4< $10
053 53 00           jms $300              ;fetch the pseudo instruction code into R2R3
055 51 00           jms $100              ;execute the associated routine
057 71 5a jcnb_047: isz 1 $05a            ;inc R0R1, pseudo code instruction pointer
059 60              inc 0
05a 14 4b iszf_05a: jcn AZ $04b           ;jump back, if ACC returned by the pseudo instruction
was 0
05c f7              tcc
05d 14 57           jcn AZ $057           ;if CY returned by the pseudo instruction was 0, R0R1 is
incremented again
                                          ;(the jump address is skipped)
05f 43 02           jun $302              ;if CY was set to 1, implement it as a pseudo code jump
instruction...


061 d4    jcnb_061: ldm 4                 ;piece of code, executed when no row is active in the
actual column of keyboard matrix
```

```
062 40 d4            jun $0d4             ;4 is the number of buttons in one column

;       i4003 shift register handling
;
;       bit0=keyboard matrix shifter clock
;       bit1=shifter data
;       bit2=printer hammer shifter clock

064 d3    subr_064: ldm 3                  ;shift high bit into keyboard shifter (Clock=1, Data=1)
065 29    subr_065: src 4<                 ;R8R9 selects ROM0
066 e2              wrr                    ;assert shifter
067 d0              ldm 0                  ;Clock=0, Data=0
068 e2    jcnb_068: wrr                    ;assert shifter
069 c0              bbl 0

;
;Synchronization with the spinning printer drum. Called strictly after the sector signal becomes
inactive. Increment R12, the
;printer sector counter. Wait for a short time, and check the state of the index signal. If it
is active, clear R12.
;

06a 6c    subr_06a: inc 12                 ;R12, the printer drum sector counter is incremented
06b 22 20           fim 1< $20
06d 23    iszb_06d: src 1<
06e ea              rdr                    ;read ROM2 input port
06f f6              rar                    ;index signal is rotated into CY
070 73 6d           isz 3 $06d             ;jump back 15 times (short wait)
072 1a 76           jcn C0 $076            ;jump, if index signal is inactive
074 f0              clb
075 bc              xch 12                 ;clear R12, the printer drum sector counter
076 c0    jcnf_076: bbl 0

;
;piece of code for the keyboard matrix handling, buffer clearing, when two buttons are pressed
at the same time
;

077 a9    jcnb_077: ld 9                   ;check the status of the current row
078 14 d9           jcn AZ $0d9            ;go back to the next row, if no button is pressed in
this column
                                           ;continue, if two buttons are simultaneously pressed in
different columns
07a 28 00 jxnb_07a: fim 4< $00             ;clear the keyboard buffer
07c f0              clb
07d 51 4a           jms $14a               ;initialize the keyboard buffer (clear KR.M0-F, KR.S0-1)
07f 40 f7           jun $0f7               ;jump to exit from keyboard handling

;
;       Keyboard decode table for translating the keyboard scan code into function code and
parameter
;
;       upper half byte=parameter
;       lower half byte=function code
;

081 bb         ;CM
082 c7         ;RM
083 63         ;M-
084 53         ;M+
085 19         ;SQRT
086 1a         ;%
087 68         ;M=-
088 58         ;M=+
089 05         ;diamond
08a 41         ;/
```

```
08b 31          ;*
08c 18          ;=
08d 22          ;-
08e 12          ;+
08f 05          ;another diamond
090 0c          ;000
091 9d          ;9
092 6d          ;6
093 3d          ;3
094 bd          ;.
095 8d          ;8
096 5d          ;5
097 2d          ;2
098 06          ;00
099 7d          ;7
09a 4d          ;4
09b 1d          ;1
09c 0d          ;0
09d ad          ;S
09e a4          ;EX
09f 0e          ;CE
0a0 bf          ;C

;
;       table for translating the function code into pseudo code entry address
;
;Note: table theoretically is started at address 0a0, but the first entry is not used

0a1 06          ;div/mul
0a2 91          ;+/-
0a3 98          ;M+/M-
0a4 f1          ;Ex
0a5 cd          ;diamond
0a6 d7          ;00
0a7 fd          ;RM
0a8 8a          ;=,M=+/M=-
0a9 05          ;Sqrt
0aa 61          ;%
0ab f9          ;CM
0ac d7          ;000
0ad d7          ;digit
0ae ca          ;CE
0af c5          ;C

;-------------------------------------------------------------------------------------------------
-----------------------------------
;Keyboard handling
;
;This part checks the first 8 columns of the keyboard matrix, and calculates the scan code based
on the position of the button in
;the matrix. When a button is pressed, the scan code is placed into the keyboard buffer stored
in KR. When two buttons are pressed
;or held down simultaneously, the buffer is cleared.
;
;This is synchronized to the printer drum rotation and is called strictly after checking the
sector signal (TEST pin of CPU).
;Typically after a "lback1: jcn TZ lback1" loop, so the sector signal just became inactive, and
terminated after a
;lback2: jcn TN lback2 loop, when the sector signal becomes active.
;-------------------------------------------------------------------------------------------------
-----------------------------------

0b0 50 6a subr_0b0: jms $06a            ;R12 synchronization with the printer drum sectors
0b2 28 07           fim 4< $07
0b4 50 64 iszb_0b4: jms $064            ;shift one high bit into keyboard shifter
0b6 79 b4           isz 9 $0b4          ;loop back, (gives 9 pulses, deactivates the entire
```

```
keyboard shifter except last column)
0b8 26 18          fim 3< $18          ;R6=1 for selecting ROM1, R7=loop counter (16-8=8
columns are checked)
0ba 22 00          fim 1< $00          ;Clear R2 and R3, scan code counter
0bc d1             ldm 1
0bd 50 65          jms $065            ;shift one low bit into keyboard shifter (select the
first column, other columns are high)

0bf 27    iszb_0bf: src 3<
0c0 ea             rdr                 ;Read ROM1 port, rows of the selected keyboard column
0c1 fc             kbp                 ;Decode the lines (0->0, 1->1, 2->2, 4->3, 8->4, rest-
>15)
0c2 b9             xch 9               ;place the code into R9
0c3 a2             ld 2
0c4 f5             ral                 ;R2 bit3 is shifted into CY, highest bit of possible
scan code
0c5 f7             tcc
0c6 1c 77          jcn AN $077         ;jump, if a pressed button has already been collected
(and may continue at $0d9)
0c8 a9             ld 9
0c9 79 cd          isz 9 $0cd          ;inc R9, and jump, if maximum one column is active
0cb 40 7a          jun $07a            ;jump to clear the buffer and exit from the keyboard
processing
                                       ;(two buttons are pressed in the same column)
0cd 14 61 iszf_0cd: jcn AZ $061        ;jump, if none of the lines are active (ACC=4, and
continue at $0d4)
0cf b2             xch 2
0d0 f5             ral
0d1 fa             stc
0d2 f6             rar
0d3 b2             xch 2               ;R2.bit3 is set to high (indicating, that a button is
pressed)
0d4 83    junf_0d4: add 3              ;ACC=1..4, if line is decoded, or 4, if no line is
active
0d5 b3             xch 3               ;adding ACC to scan code counter, R3=lower half
0d6 d0             ldm 0
0d7 82             add 2               ;adding carry to the upper half
0d8 b2             xch 2
0d9 50 64 jcnf_0d9: jms $064           ;shift one high bit into keyboard shifter (select the
next column in the matrix)
0db 77 bf          isz 7 $0bf          ;loop back, check the next columns of the matrix

0dd 29             src 4<              ;select the keyboard buffer
0de a2             ld 2                ;R2.bit3 indicates, if a button is pressed
0df f5             ral
0e0 f7             tcc
0e1 14 f8          jcn AZ $0f8         ;jump, if no button is pressed (clear the keyboard
pressing status)
0e3 ef             rd3                 ;check KR.S3, the keyboard pressing status
0e4 f2             iac                 ;ACC=1,CY=0 (when KR.S3=15) or ACC=0,CY=1 (when KR.S3=0)
0e5 f7             tcc                 ;ACC=0 or ACC=1
0e6 1c f7          jcn AN $0f7         ;jump, if the keyboard pressing status is 15 (a button
is held down)
0e8 ec             rd0                 ;a button is pressed right now, it should be placed into
the keyboard buffer
0e9 b9             xch 9               ;R9=KR.S0, the keyboard buffer pointer
0ea 29             src 4<
0eb a3             ld 3
0ec e0             wrm                 ;write R3 (lower half of the scan code) into the buffer
0ed 69             inc 9
0ee 29             src 4<
0ef e9             rdm                 ;read next byte, and if it is not 0, then
0f0 1c 7a          jcn AN $07a         ;jump to clear the buffer and exit from the keyboard
processing (overrun case)
0f2 a2             ld 2
0f3 e0             wrm                 ;write R2 (upper half of the scan code) into the buffer
```

```
0f4 69              inc 9
0f5 a9              ld 9
0f6 e4              wr0                 ;KR.S0=R9 -> store the incremented buffer pointer
0f7 df    jxnf_0f7: ldm 15              ;KR.S3=15 -> a button is held down
0f8 e7    jcnf_0f8: wr3                 ;write the keyboard pressing status
0f9 28 00           fim 4< $00          ;exit from the keyboard check, initialize R6R7 -> WR,
R8R9 -> KR
0fb 26 10           fim 3< $10
0fd 19 fd jcnb_0fd: jcn TN $0fd         ;wait for the active printer drum sector signal
0ff c0              bbl 0

100 33    subr_100: jin 1<              ;jump to the pseudo instruction code associated routine

;
;       Store the working register into another register.
;
;BPC_01:        MOV IR,WR
;BPC_02:        MOV CR,WR
;BPC_03:        MOV RR,WR
;BPC_04:        MOV DR,WR

101 a5    vmbc_101: ld 5                ;target=IR (function code+4), load function code into
ACC
102 f2    vmbc_102: iac                 ;target=CR
103 f2    vmbc_103: iac                 ;target=RR
104 86    vmbc_104: add 6               ;target=DR
105 b8              xch 8               ;source and destination is exchanged
106 b6              xch 6
107 41 0e           jun $10e            ;jump to copy numbers

;
;       Load the content of a register into the working register
;
;BPC_09:        MOV WR,MR
;BPC_0A:        MOV WR,TR
;BPC_0B:        MOV WR,SR
;BPC_0C:        MOV WR,CR
;BPC_0D:        MOV WR,RR
;BPC_0E:        MOV WR,DR

109 66    vmbc_109: inc 6               ;source=MR
10a 66    subr_10a: inc 6               ;source=TR
10b 66    vmbc_10b: inc 6               ;source=SR
10c 66    vmbc_10c: inc 6               ;source=CR
10d 66    vmbc_10d: inc 6               ;source=RR
10e 27    junf_10e: src 3<              ;source=DR,  move number into another number,
NR(R8)=NR(R6)
10f e9    vmbc_10f: rdm
110 29    vmbc_110: src 4<
111 e0              wrm                 ;number is moved digit by digit
112 69              inc 9
113 77 0e           isz 7 $10e          ;loop for all digits

115 27              src 3<              ;copy status character 0-1
116 ec              rd0                 ;plus/minus sign
117 b3              xch 3
118 ed              rd1                 ;place of digit point
119 29              src 4<
11a e5              wr1
11b b3              xch 3               ;R3=place of plus/minus sign
11c e4    vmbc_11c: wr0
11d c0              bbl 0

;
;       Adding two numbers
;
```

```
;BPC_1E:         ADD IR,WR
;BPC_21:         ADD DR,WR

11e d4    vmbc_11e: ldm 4                 ;target=IR      (function code + 4)
11f 85              add 5
120 b6              xch 6

121 29    iszb_121: src 4<                ;NR(R6)=NR(R6)+NR(R8), two numbers are added digit by
digit
122 e9              rdm
123 27              src 3<
124 eb              adm                   ;adding and daa correcting one digit
125 fb              daa
126 e0              wrm
127 69              inc 9
128 77 21           isz 7 $121            ;loop for all digits
12a f1              clc
12b c0              bbl 0

;
;       Subtracting two numbers
;
;BPC_2C:         SUB WR,IR, jump, if result is not negative (R13 is incremented at jump)
;BPC_31:         SUB IR,WR, jump, if result is not negative (R13 is incremented at jump)
;BPC_34:         SUB DR,WR, jump, if result is not negative (R13 is incremented at jump)

12c d4    vmbc_12c: ldm 4
12d 85              add 5
12e b8              xch 8                 ;source is set to function code + 4
12f 41 33           jun $133              ;target is set to 1

131 d4    junf_131: ldm 4
132 85              add 5
133 b6    junf_133: xch 6                 ;target is set to function code + 4

134 fa    vmbc_134: stc                   ;NR(R6)=NR(R6)-NR(R8), two numbers are subtracted digit
by digit
135 f9    iszb_135: tcs                   ;ACC=9+CY (10 or 9), CY=0
136 29    vmbc_136: src 4<
137 e8              sbm                   ;ACC=10(9)-NR(R8).M(R9)
138 f1              clc
139 27              src 3<
13a eb              adm                   ;ACC=NR(R6).M(R7)+(10(9)-NR(R8).M(R9))
13b fb              daa
13c e0    vmbc_13c: wrm                   ;NR(R6).M(R7)=daa adjusted result
13d 69              inc 9
13e 77 35           isz 7 $135            ;loop for all digits

140 1a 43 vmbc_140: jcn C0 $143           ;skip R13 incrementing, if last digit does not generate
carry
142 6d              inc 13
143 c1    jcnf_143: bbl 1                 ;prepare pseudo code jump

;
; clear a register including status character 0 and 1
;
;BPC_44:         CLR MR  (MR=0)
;BPC_45:         CLR TR  (TR=0)
;BPC_46:         CLR SR  (SR=0)
;BPC_47:         CLR CR  (CR=0)
;BPC_48:         CLR RR  (RR=0)
;BPC_49:         CLR DR  (DR=0)
;BPC_4A:         CLR WR  (WR=0)

144 68    vmbc_144: inc 8                 ;target=MR
145 68              inc 8                 ;target=TR
```

```
146 68    subr_146: inc 8                 ;target=SR
147 68    vmbc_147: inc 8                 ;target=CR
148 68    vmbc_148: inc 8                 ;target=RR
149 68    subr_149: inc 8                 ;target=DR

14a 29    subr_14a: src 4<                ;NR(R8).M(R9)=ACC (=0)
14b e0              wrm                   ;clearing the number digit by digit
14c 79 4a           isz 9 $14a            ;loop for all digits
14e e4              wr0                   ;clear sign
14f e5              wr1                   ;clear place of digit point
150 c0              bbl 0

;
;       On digit left shift. The number is shifted through R13
;
;BPC_51:        SHL RR  one digit left shift of RR with R13
;BPC_52:        SHL DR  one digit left shift of DR with R13
;BPC_53:        SHL WR  one digit left shift of WR with R13

151 68    vmbc_151: inc 8                 ;target=RR
152 68    vmbc_152: inc 8                 ;target=DR

153 29    iszb_153: src 4<
154 e9              rdm                   ;load current digit into ACC
155 bd              xch 13                ;previous and current digit is exchanged between ACC and
R13
156 e0              wrm                   ;save the previous digit
157 79 53           isz 9 $153            loop for next digits
159 c0              bbl 0


;
;       On digit right shift. The number is shifted through R13
;
;
;BPC_5A:        SSR RR  one digit right shift of 14 digit length RR with R13 (R13 is shifted
into digit 14)
;BPC_5D:        SHR RR  one digit right shift of RR with R13 (0 is shifted from right)
;BPC_5E:        SHR DR  one digit right shift of DR with R13 (0 is shifted from right)
;BPC_5F:        SHR WR  one digit right shift of WR with R13 (0 is shifted from right)

15a de    vmbc_15a: ldm 14                ;only 14 digits are shifted
15b b9              xch 9
15c ad              ld 13
15d 68    vmbc_15d: inc 8                 ;target=RR
15e 68    vmbc_15e: inc 8                 ;target=DR

15f bd    subr_15f: xch 13                ;one digit right shift of NR(R8).M(R9) with R13
160 a9              ld 9
161 f8    jcnb_161: dac                   ;decrement R9, loop counter
162 f1              clc
163 b9              xch 9
164 29              src 4<
165 e9              rdm                   ;load current digit into ACC
166 bd              xch 13                ;previous and current digit is exchanged between ACC and
R13
167 e0              wrm                   ;save the previous digit
168 a9              ld 9
169 1c 61           jcn AN $161           ;loop for next digits
16b c0              bbl 0

;
;       checking, whether status character 2 of certain RAM register is 0. CY=1, if it is 0
;
;BPC_6C:        JPC MODENN      jump, if RR.S2=0
;BPC_6D:        JPC MOPN        jump, if DR.S2=0
```

```
;BPC_6E:         JPC NTRUNC      jump, if WR.S2=0

16c 68    vmbc_16c: inc 8               ;source=RR
16d 68    vmbc_16d: inc 8               ;source=DR
16e 29    vmbc_16e: src 4<              ;source=WR
16f ee              rd2                 ;read status character 2
170 f8              dac                 ;decrement, only 0->15 leaves CY=0
171 f3              cmc                 ;complement carry, the pseudo jump condition
172 c1              bbl 1               ;prepare pseudo code jump

;
;       read bit 0 of status character 2 of certain RAM into CY
;
;BPC_73:         JPC OVFL        jump, if SR.S2.bit0>0
;BPC_74:         JPC MENTDP      jump, if CR.S2.bit0>0
;BPC_75:         JPC MODEMD      jump, if RR.S2.bit0>0
;BPC_76:         JPC MOPMUL      jump, if DR.S2.bit0>0
;BPC_77:         JPC ROUND       jump, if WR.S2.bit0>0

173 68    subr_173: inc 8               ;source=SR
174 68    vmbc_174: inc 8               ;source=CR
175 68    vmbc_175: inc 8               ;source=RR
176 68    vmbc_176: inc 8               ;source=DR
177 29    vmbc_177: src 4<
178 ee              rd2                 ;read status character 2
179 f6              rar                 ;rotate bit 0 into carry, the pseudo jump condition
17a c1              bbl 1               ;prepare pseudo code jump

;
;       read bit 3 of status character 2 of certain RAM into CY
;
;BPC_7B:         JPC MOPCONST    jump, if DR.S2.bit3>0

17b 27    vmbc_17b  src 3<
17c ee    vmbc_17c: rd2                 ;read status character 2
17d f5              ral                 ;rotate bit 3 into CY, the pseudo jump condition
17e c1              bbl 1               ;prepare pseudo code jump

;
;       clear status character 2 of certain number
;
;BPC_7F:         CLR OVFL        clear SR.S2
;BPC_82:         CLR MOP         clear DR.S2

17f 66    vmbc_17f: inc 6               ;target=SR
180 66    subr_180: inc 6               ;target=CR
181 66    junb_181: inc 6               ;target=RR
182 27    junb_182: src 3<              ;target=DR
183 e6              wr2                 ;write status character 2 of target (in fact it is
cleared as ACC=0)
184 c0              bbl 0

;
;       set status character 2 to a value
;
;BPC_85:         SET OVFL        SR.S2=1, set overflow
;BPC_86:         SET MENTDP      CR.S2=1, set that number is entered with digit point
;BPC_87:         SET MODEMD      RR.S2=1, set that number is used for mul/div operation
;BPC_8A:         SET MODEAS      RR.S2=8, set that number is used for add/sub operation
;BPC_8D:         SET MOPPAR      DR.S2=function parameter, set the multiplication/division from
function parameter
;BPC_90:         SET MOPCONST    DR.S2.bit3=1, set that multiply/divide operation is with
constant value

185 66    vmbc_185: inc 6               ;target=SR
186 66    vmbc_186  inc 6               ;target=CR
```

```
187 d1    vmbc_187: ldm 1                 ;target=RR
188 41 81           jun $181              ;set NR(R6+1).S2=1

18a d8    vmbc_18a: ldm 8
18b 41 81           jun $181              ;set NR(R6+1).S2=8

18d a4    vmbc_18d: ld 4                  ;ACC = parameter
18e 41 82           jun $182              ;set NR(R6).S2=parameter

190 27    vmbc_190: src 3<
191 ee              rd2                   ;set high bit of NR(R6).S2 to 1
192 f5              ral
193 fa              stc
194 f6              rar
195 e6              wr2
196 c0              bbl 0

;
;       checking, whether the number contains any nonzero digit
;
;BPC_97:        JPC NBIG_IR     jump, if digits 14-15 of IR does not contain any value
;BPC_9A:        JPC NBIG_WR     jump, if digits 14-15 of WR does not contain any value
;BPC_9E:        CLR DIGIT + JPC NBIG_DR         jump, if digits 14-15 of DR does not contain any
value. R13=0
;BPC_A0:        CLR DIGIT + JPC ZERO_DR         clear R13 and jump, if DR does not contain any
value
;BPC_A2:        JPC ZERO_WR     jump, if WR does not contain any value

197 d4    vmbc_197: ldm 4
198 85              add 5                 ;ACC=function code or function code + 4
199 b8              xch 8                 ;R8 points to IR

19a de    vmbc_19a: ldm 14
19b b9              xch 9                 ;R9=14
19c 41 a2           jun $1a2

19e de    vmbc_19e: ldm 14
19f b9              xch 9                 ;R9=14 and ACC=previous R9 (=0)

1a0 bd    subr_1a0: xch 13                ;save ACC=0 into R13
1a1 68              inc 8

1a2 29    subr_1a2: src 4<                ;check whether the number contains any digit. Return
jump with CY=1, if the number is empty
1a3 df    vmbc_1a3: ldm 15
1a4 eb              adm                   ;number is added in binary mode to the maximum value
digit by digit
1a5 79 a2           isz 9 $1a2            ;loop for the rest of digits

;BPC_A7:        JMP     Unconditional jump

1a7 f3    vmbc_1a7: cmc                   ;negate the pseudo jump condition
1a8 c1              bbl 1                 ;prepare pseudo code jump

;BPC_A9:        JPC BIG_DIGIT   Jump, if R13>9

1a9 ad    vmbc_1a9: ld 13                 ;load R13
1aa fb    vmbc_1aa: daa                   ;set CY=1, the pseudo jump condition, if R13>9
1ab c1    vmbc_1ab: bbl 1                 ;prepare pseudo code jump

;BPC_AC:        JPC ZERO_DIGIT + DEC DIGIT      decrement R13 and jump, if R13 was 0 before the
decrement

1ac ad    vmbc_1ac: ld 13
1ad f8              dac                   ;ACC=decremented R13, will be placed back to R13
```

```
;BPC_AE:          CLR DIGIT + JMP          clear R13 and jump

1ae bd    vmbc_1ae: xch 13
1af f3              cmc                    ;negate the pseudo jump condition
1b0 c1              bbl 1                  ;prepare pseudo code jump

;BPC_B1:          JPC NEWOP        jump, if function code < 8 (new add/sub/mul/div operation)

1b1 d7    vmbc_1b1: ldm 7
1b2 95              sub 5                  ;R5=function code; set CY=1, the pseudo jump condition,
if R5<8
1b3 c1              bbl 1                  ;prepare pseudo code jump

;BPC_B4:          JPC MEMOP        jump, if function parameter > 3 (new memory operation)

1b4 dc    vmbc_1b4: ldm 12
1b5 84              add 4                  ;R4=function parameter; set CY=1, the pseudo jump
condition, if R4>3
1b6 c1              bbl 1                  ;prepare pseudo code jump

;BPC_B7:          JPC ROTFC        rotate the function code one bit right, jump if the next bit is
0

1b7 a5    vmbc_1b7  ld 5                   ;rotate R5=function code with 1 bit right
1b8 f6              rar                    ;bit 0 is rotated to CY
1b9 b5              xch 5                  ;rotated value is saved back
1ba f3              cmc                    ;complement CY, the pseudo jump condition
1bb c1              bbl 1                  ;prepare pseudo code jump

;BPC_BC:          JPC ODDPAR       jump, if bit0 of parameter>0

1bc a4    vmbc_1bc: ld 4                   ;load R4=parameter into ACC
1bd f6    vmbc_1bd: rar                    ;rotate bit 0 into CY, the pseudo jump condition
1be c1              bbl 1                  ;prepare pseudo code jump

;BPC_BF:          SET DP_IR        set digit point place of indirect register (IR.S1=R11)

1bf d4    vmbc_1bf: ldm 4
1c0 85              add 5                  ;ACC=function code + 4
1c1 b8              xch 8                  ;set it to target register

;BPC_C2:          SET DP_WR        set digit point place of working register (WR.S1=R11)

1c2 29    vmbc_1c2: src 4<
1c3 ab              ld 11
1c4 e5              wr1                    ;write place of digit point
1c5 c0              bbl 0

;BPC_C6:          GET DP_WR        get digit point place of working register (R11=WR.S1)

1c6 29    vmbc_1c6: src 4<
1c7 ed              rd1                    ;read place of digit point
1c8 bb              xch 11
1c9 c0              bbl 0

;BPC_CA:          INC DPCNT        increment digit point counter (increment R10R11)

1ca 7b cd vmbc_1ca: isz 11 $1cd            ;increment lower part, jump, if not zero
1cc 6a              inc 10                 ;increment upper part
1cd c0    iszf_1cd: bbl 0

;BPC_CE:          JPC NBIG_DPCNT   jump, if R10R11<14
;BPC_CF:          JPC ZERO_DPCNT   jump, if R10R11=0

1ce dd    vmbc_1ce: ldm 13
1cf 9b    vmbc_1cf: sub 11                 ;subtract the lower part from 13
```

```
1d0 f3              cmc
1d1 d0              ldm 0               ;subtract the upper part from 0
1d2 9a              sub 10              ;pseudo jump condition is set at no borrow
1d3 c1    vmbc_1d3: bbl 1               ;prepare pseudo code jump

;Pseudo instruction code jump table. Normally pseudo instruction code execution can be directly
started on address range $100-$1ff.
;This is a jump table to functions, which are implemented on other pages

1d4 42 d3 vmbc_1d4: jun $2d3            ;BPC_D4: jump, if WR and IR have different sign
1d6 00              nop
1d7 42 94 vmbc_1d7: jun $294            ;BPC_D7: digit functions
1d9 42 a3 vmbc_1d9: jun $2a3            ;BPC_D9: WR=TR, clear SR & TR; recall main total
1db 42 aa vmbc_1db: jun $2aa            ;BPC_DB: Set function code=3, and jump
1dd 42 ae vmbc_1dd: jun $2ae            ;BPC_DD: decrement R10R11
1df 42 b3 vmbc_1df: jun $2b3            ;BPC_DF: WR.S1=WR.S3, R10R11=difference between required
an actual digit point
1e1 42 b9 vmbc_1e1: jun $2b9            ;BPC_E1: digit point counter adjust for division
1e3 42 ca vmbc_1e3: jun $2ca            ;BPC_E3: digit point counter adjust for multiplication
1e5 42 de vmbc_1e5: jun $2de            ;BPC_E5: Sign of result register for multiplication or
division + R13=15
1e7 42 e7 vmbc_1e7: jun $2e7            ;BPC_E7: complement WR.S0 (change the sign of WR)
1e9 42 ec vmbc_1e9: jun $2ec            ;BPC_E9: rounding, if R13>4 then increment WR (and R14
too)
1eb 42 46 vmbc_1eb: jun $246            ;BPC_EB: end of printing with advancing the paper and
R10R11=0, R14R15=0
1ed 44 00 vmbc_1ed: jun $400            ;BPC_ED: square root (optional)

;BPC_EF: CLR MENT + CLR OVFL + RET  clear CR.S2, SR.S2, TR.S2 and exit
;BPC_F1: CLR MODE + CLR MENT + RET      clear RR.S2, CR.S2 and exit
;BPC_F3: CLR MODE + RET         clear RR.S2 and exit

1ef 51 80 vmbc_1ef: jms $180            ; R6=R6+2, clear status character 2 of NR(R6)
1f1 51 81 vmbc_1f1: jms $181            ; R6=R6+1, clear status character 2 of NR(R6)
1f3 51 81 vmbc_1f3: jms $181            ; R6=R6+1, clear status character 2 of NR(R6)
1f5 2a 00           fim 5< $00
1f7 40 00           jun $000            ;exit from the pseudo code interpreter

;BPC_F9..FF: Printing functions:

;BPC_F9:        PRN FPAR,C      print number with function parameter and char=11 "C" in last
column (not used)
;BPC_FA:        PRN FPAR,MEM    print number with function parameter and char=12 "M" in last
column
;BPC_FB:        PRN FPAR,FCODE  print number with function parameter and empty character in last
column
;BPC_FC:        PRN FPAR        print number with function parameter and empty character in last
column
;BPC_FD:        PRN ROUND,FPAR  print number with optional rounding char and function parameter
in last column
;                               (determined by R14.bit0: 0=empty, 1=code 7 (rounding up char))
;BPC_FE:        PRN FCODE       print number with function code and empty character in last
column
;BPC_FF:        PRN OVFL        print unimplemented number (dots with empty extra columns)

1f9 6f              inc 15              ; (R15 will be 9)
1fa 6f    vmbc_1fa: inc 15              ; (R15 will be 10)
1fb 6f    vmbc_1fb: inc 15              ; (R15 will be 11)
1fc 6f    vmbc_1fc: inc 15              ; (R15 will be 12)
1fd 6f    vmbc_1fd: inc 15              ; (R15 will be 13)
1fe 6f    vmbc_1fe: inc 15              ; (R15 will be 14)
1ff bf    vmbc_1ff: xch 15              ; (R15 will be 15)
200 f4              cma
201 bf              xch 15              ; R15 is complemented

;setting the printing method, determined by the value in R15 (and R14=rounding)
```

```
202 7f 10           isz 15 $210
                                        ;R15 was 15: unimplemented number (overflow/divide by 0)
204 da              ldm 10              ;load 10 (code of digit point)
205 51 4a           jms $14a            ;fill WR with 10s (WR.S0 too: positive number, WR.S1
too: not used)
207 2e ff           fim 7< $ff          ;R14R15=$FF: last two columns will be empty
209 ba              xch 10              ;R10=10: "place of digit point" would generate a point
too
20a df              ldm 15
20b b9              xch 9               ;R9=15: 14 valid character
20c 29              src 4<
20d e0              wrm                 ;WR.M15=0
20e 42 2c           jun $22c            ;jump to start the printing

210 7f 17 iszf_s10: isz 15 $217
                                        ;R15 was 14: number with function code and empty
character in last column
212 df              ldm 15
213 bf              xch 15              ;R15=15 (empty column)
214 a5              ld 5                ;function code
215 42 26           jun $226            ;jump to save ACC into R14

217 d1    iszf_217: ldm 1
218 8f              add 15
219 f7              tcc
21a 14 25           jcn AZ $225         ;jump, if R15<13
                                        ;      number with function parameter and a character
(can be empty) in the last column

                                        ;R15 was 13: number with optional rounding char and
function parameter in the last column
21c a4              ld 4
21d bf              xch 15              ;R15=function parameter
21e be              xch 14              ;ACC=R14, set previously by the rounding (0=truncating,
1=rounding up)
21f f6              rar                 ;CY=R14.bit0
220 f3              cmc                 ;CY=complement of R14.bit0
221 de              ldm 14
222 f6              rar                 ;ACC=8*CY+7  (7=rounding up char, 15=empty char)
223 42 26           jun $226            ;jump to save ACC into R14

225 a4    jcnf_225: ld 4                ;load parameter into R14
226 be    junf_226: xch 14              ;save ACC into R14, code of character in last column
227 29              src 4<
228 ed              rd1
229 ba              xch 10              ;R10=place of digit point
22a ed              rd1
22b bb              xch 11              ;R11=place of digit point

22c 11 2c jcnb_22c: jcn TZ $22c         ;wait for the inactive printer drum sector signal
22e d2              ldm 2
22f bd              xch 13              ;R13=2
230 ec              rd0                 ;read WR.S0 (sign)
231 f6              rar
232 f7              tcc                 ;ACC=0 (WR positive) or 1 (WR negative)
233 e1              wmp                 ;switch the printing color into red in case of WR has
minus sign, output to RAM0 port
234 50 b0           jms $0b0            ;keyboard handling
236 68              inc 8               ;R8R9 points to WR again (keyboard handling puts it to
KR)
                                        ;R6R7 points to WR too

237 6b    jcnb_237: inc 11              ;search for the place of the first digit before the
digit point, result in R11
238 ab              ld 11
```

```
239 b9              xch 9              ;R9=points to part, to be checked (started from place of digit point + 1)
23a 51 a2           jms $1a2           ;check, whether the remaining part of the number contains any digit
23c f7              tcc
23d 14 37           jcn AZ $237        ;jump back, if the remaining part of the number is not empty

;by this point:
; R6=1 (select WR)
; R7=0 (used as a digit loop counter)
; R8=1 (select WR)
; R9=0
; R10=place of digit point
; R11=place of first nonzero digit before the digit point+1
; R13=2 (used as a printer sector loop counter)
; R14=character code on column before the last column (or 13..15, if that is empty)
; R15=character code on the last column (or 13..15, if that is empty)

;printing: R13 loop counter for the printer sectors

23f 11 3f junb_23f: jcn TZ $23f        ;wait for the inactive printer drum sector signal
241 f0              clb
242 e1              wmp                ;printer control signals are set to inactive
243 e2              wrr
244 7d 53           isz 13 $253        ;jump to next sector, if there is

;BPC_EB:        PRN ADVANCE + CLR DPCNT         end of printing with advancing the paper and R10R11=0, R14R15=0

246 2a 0c subr_246: fim 5< $0c         ;R10R11=$0C
248 2e 00           fim 7< $00         ;R14R15=$00
24a d8              ldm 8
24b 11 4b jcnb_24b: jcn TZ $24b        ;wait for the inactive printer drum sector signal
24d e1              wmp                ;Write RAM0 port, first 8, later 3 times 0 (advance the printer paper with a line)
24e 50 b0           jms $0b0           ;Keyboard handling
250 7b 4b           isz 11 $24b        ;loop back
252 c0              bbl 0

253 50 6a iszf_253: jms $06a           ;R12 synchronization with the printer drum sectors
255 b8              xch 8              ;clear R8

;printing: R7 loop for the digits - filling the printer shifter for one sector

256 dd              ldm 13             ;(if R15=13, then the number is empty)
257 9f              sub 15             ;ACC=13-R15
258 f1              clc
259 1c 5f           jcn AN $25f        ;jump, if R15<>13
25b ba              xch 10             ;R10=0 (if R15=13, empty columns are printed)
25c df              ldm 15             ;(handling of empty columns)
25d 42 61           jun $261
25f 27    jcnf_25f: src 3<             ;(handling of valid digits)
260 e9              rdm                ;read one digit into ACC
261 77 77           isz 7 $277         ;jump to next digit, if there is still

263 aa              ld 10              ;pattern of extra two columns are fetched from R14 and R15
264 1c 68           jcn AN $268        ;jump, if R10<>0 (digit point is already shifted)
266 52 8f           jms $28f           ;shift one inactive column into printer shifter (CY=0)
268 af    jcnf_268: ld 15
269 52 8a           jms $28a           ;if R15=R12, shift 1 into printer shifter else shift 0
26b ae              ld 14
26c 52 8a           jms $28a           ;if R14=R12, shift 1 into printer shifter else shift 0

26e 19 6e jcnb_26e: jcn TN $26e        ;wait for the active printer drum sector signal
```

```
270 d2               ldm 2
271 29               src 4<
272 e1               wmp                  ;fire printer hammers
273 50 b2            jms $0b2             ;Keyboard handling (R7 is cleared!)
275 42 3f            jun $23f             ;loop back for the next sectors

277 52 8a iszf_277: jms $28a             ;if ACC=R12, shift 1 into printer shifter else shift 0
279 aa               ld 10
27a 14 83            jcn AZ $283          ;jump, if R10=0 (there is no digit point)
27c 97               sub 7
27d f1               clc
27e 1c 83            jcn AN $283          ;jump, if R10<>R7 (digit point is not in this position)

280 da               ldm 10               ;shift the digit point into the shifter
281 52 8a            jms $28a             ;if R12=10, shift 1 into printer shifter else shift 0

283 a7    jcnf_283: ld 7                 ;check, whether the loop counter exceeded the number of
valid digits
284 9b               sub 11
285 f7               tcc
286 14 56            jcn AZ $256          ;loop back for the next valid digits
288 42 5c            jun $25c             ;loop back for the empty columns

28a 9c    subr_28a: sub 12               ;if ACC=R12, shift 1 into printer shifter else shift 0
28b f1               clc
28c 1c 8f            jcn AN $28f
28e fa               stc
28f d1    subr_28f: ldm 1                ;shift CY into printer shifter
290 f5               ral
291 f5               ral                  ;ACC=4+2*CY
292 40 65            jun $065             ;shift one low bit into printer shifter

;BPC_D7:        DIGIT   this function is called, when a digit, "00", "000", digit point or minus
sign button is pressed

294 a4    junf_294: ld 4
295 bd               xch 13               ;R13=digit
296 26 40            fim 3< $40
298 27               src 3<
299 ee               rd2                  ;read CR.S2, digit entry mode status
29a 1c a1            jcn AN $2a1          ;Jump, if the calculator is already in digit entry mode
29c d8               ldm 8
29d e6               wr2                  ;put 8 into the digit entry mode status
29e f0               clb
29f 51 4a            jms $14a             ;clear WR, WRS0, WRS1
2a1 41 c6            jun $1c6             ;R11=WR.S1, place of digit point

;BPC_D9:        MOV WR,TR + CLR TR + CLR SR     recall main total (WR=TR, clear SR & TR)

2a3 51 0a junf_2a3: jms $10a             ;WR=TR
2a5 51 46            jms $146             ;clear SR (including S0 and S1)
2a7 51 49            jms $149             ;clear TR (including S0 and S1)
2a9 c0               bbl 0

;BPC_DB:        SET MRMFUNC + JMP       set function code=3 (memory function), and jump

2aa d3    junf_2aa: ldm 3
2ab b5               xch 5                ;R5=function code is set to 3
2ac fa               stc                  ;set CY=1, the pseudo jump condition
2ad c1               bbl 1                ;prepare pseudo code jump

;BPC_DD:        DEC DPCNT               decrement R10R11

2ae d1    junf_2ae: ldm 1
2af b3               xch 3                ;R3=1
2b0 bb               xch 11               ;ACC=R11
```

```
2b1 42 c2               jun $2c2              ;jump to R10R11 adjust

;BPC_DF:        GET DPDIFF      WR.S1=WR.S3, set R10R11 to the difference between required an
actual digit point

2b3 52 f9 junf_2b3: jms $2f9              ;read the decimal places of WR and DR (R2=DR.S1,
R3=WR.S1), DR is not used
2b5 ef                  rd3                   ;read the required decimal places defined by the digit
point switch
2b6 e5                  wr1                   ;set it to WR.S1
2b7 42 c2               jun $2c2              ;jump to R10R11 adjust

;BPC_E1:        GET DPCNTDIV    digit point counter adjust for division (set R10R11 to DR.S1+
(13-R11)-WR.S1)

2b9 52 f9 junf_2b9: jms $2f9              ;read the decimal places of WR and DR (R2=DR.S1,
R3=WR.S1)
2bb dd                  ldm 13
2bc 9b                  sub 11                ;ACC=13-R11
2bd f1                  clc
2be 82                  add 2                 ;ACC=R2+(13-R11)
2bf ba                  xch 10
2c0 f7                  tcc
2c1 ba                  xch 10                ;R10=carry

; R10R11 adjust: set R10R11 to the difference between required an actual digit point
;       input:  ACC=required place of digit point
;               R3=place of digit point of the actual number

2c2 93    junf_2c2: sub 3
2c3 bb                  xch 11                ;R11=ACC-R3
2c4 f3                  cmc
2c5 ba                  xch 10
2c6 99                  sub 9                 ;borrow is subtracted from the upper half (R9=0)
2c7 ba                  xch 10                ;R10=R10-(CY)
2c8 f1                  clc
2c9 c0                  bbl 0

;BPC_E3:        GET DPCNTMUL    digit point counter adjust for multiplication
;                               set R10R11 to the sum of digital places (WR, DR and current in
R11)

2ca 52 f9 junf_2ca: jms $2f9              ;read the decimal places of WR and DR (R2=DR.S1,
R3=WR.S1)
2cc a3                  ld 3
2cd 8b                  add 11
2ce 82                  add 2
2cf bb                  xch 11                ;R11=R11+R3+R2
2d0 f7                  tcc
2d1 ba                  xch 10                ;R10=0 or 1
2d2 c0                  bbl 0

;BPC_D4:        JPC DIFF_SIGN   jump, if WR and IR have different sign (either is minus, the
other is plus)

2d3 d4    junf_2d3: ldm 4
2d4 85                  add 5
2d5 b6                  xch 6                 ;R6=function code + 4
2d6 27                  src 3<
2d7 ec                  rd0                   ;read the sign of IR
2d8 b2                  xch 2
2d9 29                  src 4<
2da ec                  rd0                   ;read the sign of WR
2db 82                  add 2                 ;bit0 of result is 0, if both number have the same sign
2dc f6                  rar                   ;rotate bit 0 into CY, the pseudo jump condition
2dd c1                  bbl 1                 ;prepare pseudo code jump
```

```
;BPC_E5:         SET DIVMUL_SIGN + MOV DIGIT,15
;       Sign of result register is set based on the WR and DR for multiplication or division
;       R13 is set to 15 for loop counting

2de 52 d6 junf_2de: jms $2d6            ;compare WR and DR sign
2e0 f7              tcc
2e1 66              inc 6               ;R6 points to RR
2e2 27              src 3<
2e3 e4              wr0                 ;set sign of RR
2e4 df              ldm 15
2e5 bd              xch 13              ;R13=15, used as "loop end" indicator at divide/multiply
2e6 c0              bbl 0

;BPC_E7:         NEG WR  complement sign of working register (change the sign of WR)

2e7 29    junf_2e7: src 4<
2e8 ec              rd0                 ;read the sign
2e9 f4              cma                 ;complement it
2ea e4              wr0                 ;write back the new sign
2eb c0              bbl 0

;BPC_E9:         ROUNDING        increment WR (and R14 too), if R13>4

2ec db    junf_2ec: ldm 11
2ed 8d              add 13              ;R13 is added to 11
2ee 1a f1           jcn C0 $2f1         ;if R13<5, CY=0, jump to add (??? jump to $2f8 would
have been better)
2f0 6e              inc 14              ;save also the fact of rounding into R14

2f1 d0    jcnf_2f1: ldm 0               ;Add CY to WR
2f2 29              src 4<
2f3 eb              adm
2f4 fb              daa                 ;add carry and decimal digit by digit
2f5 e0              wrm
2f6 79 f1           isz 9 $2f1          ;loop for the next digits
2f8 c0              bbl 0

2f9 27    subr_2f9: src 3<              ;read the decimal places of WR and DR (R2=DR.S1,
R3=WR.S1)
2fa ed              rd1
2fb b2              xch 2               ;R2=DR.S1
2fc 29              src 4<
2fd ed              rd1
2fe b3              xch 3               ;R3=WR.S1
2ff c0              bbl 0

300 32    subr_300: fin 1<              ;fetch the pseudo instruction code into R2R3 and return
301 c0              bbl 0
302 30              fin 0<              ;fetch the jump address, as the new value of pseudo code
instruction pointer into R0R1
303 40 4b           jun $04b            ;jump to the WM code interpreter

;-----------------------------------------------------------------------------------------------
-----------------------------------
;Detailed analysis of basic pseudo code list.
;-----------------------------------------------------------------------------------------------
-----------------------------------

305 fn_sqrt:   ed    ;SQRT (+ JMP num_dpadj)                        ;square root of WR is placed
into RR

306 fn_muldiv: 6c 14 ;JPC MODENN,md_prn2                            ;jump, if new number is entered
308            75 0e ;JPC MODEMD,md_prn1                            ;jump, if mul or div was the
last operation
30a            d9    ;MOV WR,TR + CLR TR + CLR SR                   ;if add or sub was the last
```

```
operation, then main total is recalled
30b             fc    ;PRN FPAR
30c             a7 0f ;JMP md_exitf
30e md_prn1:    fb    ;PRN FPAR,FCODE
30f md_exitf:   8d    ;SET MOPPAR                                 ;keep the operation (from the
parameter) for the next round
310 md_exitc:   04    ;MOV DR,WR                                  ;put the number into DR and CR
311             02    ;MOV CR,WR
312             87    ;SET MODEMD
313             ef    ;CLR MENT + CLR OVFL + RET
314 md_prn2:    fc    ;PRN FPAR
315             6d 0f ;JPC MOPN,md_exitf                          ;jump, if the other operand is
not entered yet
317             7b 0f ;JPC MOPCONST,md_exitf                      ;jump, at constant calculation
(new number for calculation)
319             76 46 ;JPC MOPMUL,mul_start                       ;jump, if previous operation is
multiply

;---------------------------------------------------------------------------------------------
-----------------------------------
;dividing:      WR <- RR = DR / WR
;
;DR and WR is left adjusted into position WR.M14<>0 and DR.M14<>0, DR is decreased by WR till it
becomes negative. WR is added back
;to DR for getting back the smallest non negative DR. The count, how many times it could be
decreased gives the next digit of
;result, which is shifted into RR. DR is shifted left for doing the subfunction for the next
digit. The same process is repeated
;14 times. Place of digit point of the result is calculated separately. Finally the result from
RR is copied to WR.
;---------------------------------------------------------------------------------------------
-----------------------------------

31b             8d    ;SET MOPPAR                                 ;divide is marked into MOP
31c div_chk0:   a2 3c ;JPC ZERO_WR,num_overf                      ;divide by zero would result
overflow
31e             48    ;CLR RR
31f             a0 73 ;CLR DIGIT + JPC ZERO_DR,num_res            ;if dividend is zero, the result
will be zero too
321             e1    ;GET DPCNTDIV                               ;digit point initialization for
divide
322 div_chkDR: 9e 32 ;CLR DIGIT + JPC NBIG_DR,div_lshDR          ;rotate DR into leftmost
position
324 div_chkWR: 9a 36 ;JPC NBIG_WR,div_lshWR                      ;rotate WR into leftmost
position
326             e5    ;SET DIVMUL_SIGN + MOV DIGIT,15             ;sign of result is set
327             51    ;SHL RR                                     ;15 is shifted into the cleared
RR, as a mark for loop end
328             51    ;SHL RR
329 div_loop:   34 29 ;SUB DR,WR + JPC NNEG,div_loop + INC DIGIT ;find, how many times the
subtraction can be done
32b             21    ;ADD DR,WR                                  ;adding back the last unneeded
subtract
32c             51    ;SHL RR                                     ;next digit of result is shifted
into RR
32d             a9 3f ;JPC BIG_DIGIT,div_finsh                    ;if shifted out number>9, end of
division
32f             52    ;SHL DR                                     ;next digit (shifted out from
RR) is shifted into DR
330             a7 29 ;JMP div_loop
332 div_lshDR:  52    ;SHL DR                                     ;one digit rotate left of DR
333             ca    ;INC DPCNT
334             a7 22 ;JMP div_chkDR
336 div_lshWR:  53    ;SHL WR                                     ;one digit rotate left of WR
337             cf 3c ;JPC ZERO_DPCNT,num_overf                   ;jump if rotate would cause
overflow
```

```
339            dd    ;DEC DPCNT
33a            a7 24 ;JMP div_chkWR

33c num_overf: ff    ;PRN OVFL                                     ;print overflow
33d            85    ;SET OVFL                                     ;set overflow flag
33e            f1    ;CLR MODE + CLR MENT + RET                    ;exit

33f div_finsh: 5d    ;SHR RR                                       ;rotate the number right

340 num_dpadj: ce 73 ;JPC NBIG_DPCNT,num_res                       ;jump, if the result contains
acceptable number of digits
342            dd    ;DEC DPCNT                                    ;otherwise shift the number to
right
343            5d    ;SHR RR                                       ;Note: the place of this
instruction could have been saved,
344            a7 40 ;JMP num_dpadj                                ;  if the jump would go back to
div_finsh

;--------------------------------------------------------------------------------------------
----------------------------------
;multiplication: WR <- RR = DR * WR
;
;As starting WR is copied to RR and DR copied to WR. DR is cleared.
;DR and RR is shifted right. Last digit of RR is placed into R13, WR is added R13 times to DR.
The process is repeated 14 times.
;Two 14 digit operand produces maximum 28 digit result. For us the most significant digits are
interesting. Therefore the 28 digit
;result is rotated towards the lower digits, till the upper 14 digits contain nonzero digits,
the place of digit point is counted
;in R10 and R11. After rotate the result is finally copied to WR.
;--------------------------------------------------------------------------------------------
----------------------------------

346 mul_start: 8d    ;SET MOPPAR                                   ;multiplication is marked in MOP
347 mul_st2:   03    ;MOV RR,WR
348            e3    ;GET DPCNTMUL                                 ;digit point initialization for
multiply
349            e5    ;SET DIVMUL_SIGN + MOV DIGIT,15               ;sign of result is set
34a            0e    ;MOV WR,DR
34b            49    ;CLR DR
34c            52    ;SHL DR                                       ;shift R13=15 into DR, but it is
immediately shifted into RR
34d mul_loopn: 5e    ;SHR DR                                       ;DR-RR is shifted right
34e            5a    ;SSR RR
34f            a9 56 ;JPC BIG_DIGIT,mul_shres                      ;jump if R13=15 was shifted out
(exit from the loop)
351 mul_loopd: ac 4d ;JPC ZERO_DIGIT,mul_loopn + DEC DIGIT         ;multiply the number with one
digit
353            21    ;ADD DR,WR                                    ;finally DR=DR+R13*WR
354            a7 51 ;JMP mul_loopd

356 mul_shres: a0 40 ;CLR DIGIT + JPC ZERO_DR,num_dpadj            ;rotate nonzero digits from DR
to RR
358            cf 3c ;JPC ZERO_DPCNT,num_overf                     ;jump if overflow occurred
35a            5e    ;SHR DR                                       ;DR-RR is shifted right
35b            5a    ;SSR RR
35c            dd    ;DEC DPCNT
35d            a7 56 ;JMP mul_shres

35f dp_mark:   86    ;SET MENTDP                                   ;digit point flag
360            f3    ;CLR MODE + RET

361 fn_percnt: fe    ;PRN FCODE
362            ca    ;INC DPCNT                                    ;increment the digit point place
counter by 2
363            ca    ;INC DPCNT
```

```
364             a7 67 ;JMP num_md

366 num_prm:    fe    ;PRN FCODE
367 num_md:     7b 6f ;JPC MOPCONST,num_mul2                      ;jump at const divide/multiply
369             90    ;SET MOPCONST
36a num_mul1:   76 47 ;JPC MOPMUL,mul_st2                         ;jump to multiply, if previous
operation is multiply
36c             02    ;MOV CR,WR                                  ;save the divisor for constant
divide
36d             a7 1c ;JMP div_chk0                               ;jump to divide
36f num_mul2:   04    ;MOV DR,WR                                  ;save the number into DR
370             0c    ;MOV WR,CR                                  ;recall previous number from CR
371             a7 6a ;JMP num_mul1                               ;jump to divide or multiply

373 num_res:    0d    ;MOV WR,RR                                  ;copy the RR result to WR
374             c2    ;SET DP_WR                                  ;set the digit point position
from R10R11
375             b1 10 ;JPC NEWOP,md_exitc                         ;jump to exit at new mul and div
operation
377             b4 7b ;JPC MEMOP,num_adj                          ;jump to adjust at M=+/M=-
379             6e 9e ;JPC NTRUNC,num_pra2                        ;jump to result print, if digit
point should not be adjusted
37b num_adj:    df    ;GET DPDIFF                                 ;WR.S1=WR.S3, set R10R11 to the
difference between required an actual digit point
                                                                  ;Rotate the number into the
required digit point place
37c num_rotl:   cf 9a ;JPC ZERO_DPCNT,num_pra1                    ;jump, if number is at the right
digit point place
37e             ce 84 ;JPC NBIG_DPCNT,num_lrot

380             ca    ;INC DPCNT                                  ;Rotate right
381             5f    ;SHR WR
382             a7 7c ;JMP num_rotl

384 num_lrot:   dd    ;DEC DPCNT                                  ;Rotate left
385             53    ;SHL WR
386             9a 7c ;JPC NBIG_WR,num_rotl
388             a7 3c ;JMP num_overf                              ;print overflow

38a fn_memeq:   6c 66 ;JPC MODENN,num_prm                         ;jump, if new number is entered
38c             75 66 ;JPC MODEMD,num_prm                         ;jump, if there is started
mul/div operation
38e             d9    ;MOV WR,TR + CLR TR + CLR SR                ;recall main total
38f             a7 98 ;JMP fn_memadd                              ;jump to add functions

;entry address at add or subtract button
391 fn_addsub:  6c 98 ;JPC MODENN,fn_memadd                       ;jump, if new number is enterer
393             75 97 ;JPC MODEMD,clr_md                          ;jump, if there is started
mul/div operation
395             a7 98 ;JMP fn_memadd                              ;jump to add functions

397 clr_md:     82    ;CLR MOP                                    ;ignore previous mul/div
operation

398 fn_memadd:  ae 7b ;CLR DIGIT + JMP num_adj                    ;jump to adjust the number to
the required digits

39a num_pra1:   b1 aa ;JPC NEWOP,num_pra3                         ;jump at new add/sub operation
39c             77 a3 ;JPC ROUND,num_round                        ;jump to rounding, if rounding
switch is in that position

39e num_pra2:   fd    ;PRN ROUND,FPAR
39f             eb    ;PRN ADVANCE + CLR DPCNT
3a0             b4 a8 ;JPC MEMOP,mem_add                          ;jump to change the function
code at M=+/M=-/M+/M-
3a2             f1    ;CLR MODE + CLR MENT + RET
```

```
3a3 num_round: e9    ;ROUNDING                                  ;do the rounding based on the
last shifted out digit in R13
3a4            9a 9e ;JPC NBIG_WR,num_pra2                      ;may generate overflow too
3a6            a7 3c ;JMP num_overf                             ;print overflow

3a8 mem_add:   db ab ;SET MEMFUNC + JMP do_prpadd               ;Set M+/M- function code

;----------------------------------------------------------------------------------------------
-----------------------------------
;add/subtract functions:
;
;By this point, numbers are shifted into the place determined by the digit point switch, thus no
shifting is needed.
;
;!!! Note, if the digit point switch is changed during an operation, the numbers are incorrectly
added/subtracted.
;
;       function code       parameter           pre1            operation1      pre2
operation2
;+               2                 1                RR=WR           TR=TR+WR        WR=RR   SR=SR+WR
;-               2                 2                RR=WR           TR=TR-WR        WR=RR   SR=SR-WR
;M+ (M=+)        3                 5                RR=WR           MR=MR+WR
;M- (M=-)        3                 6                RR=WR           MR=MR-WR
;----------------------------------------------------------------------------------------------
-----------------------------------

3aa num_pra3:  fc    ;PRN FPAR
3ab do_prpadd: c6    ;GET DP_WR

3ac do_addsub: 03    ;MOV RR,WR

3ad            bc b0 ;JPC ODDPAR,skp_neg                        ;skip negate the number at add
3af            e7    ;NEG WR                                    ;negate the number at sub
(convert it to add)
3b0 skp_neg:   d4 b7 ;JPC DIFF_SIGN,do_sub                      ;jump, when adding a negative
and a positive number

3b2            1e    ;ADD IR,WR                                 ;ADD - may generate overflow
3b3            97 bd ;JPC NBIG_IR,do_next
                                                                ;jump, if there is no overflow
3b5            31 3c ;SUB IR,WR + JPC NNEG,num_overf + INC DIGIT        ;correct back IR at
overflow and jump always

3b7 do_sub:    31 bd ;SUB IR,WR + JPC NNEG,do_next + INC DIGIT  ;SUB - never generates overflow
3b9            1e    ;ADD IR,WR
3ba            2c bc ;SUB WR,IR + JPC NNEG,do_cont              ;always goes to the next
instruction
3bc do_cont:   01    ;MOV IR,WR

3bd do_next:   0d    ;MOV WR,RR                                 ;take the original number from
RR
3be            bf    ;SET DP_IR                                 ;set the place of digit point

3bf            b4 ff ;JPC MEMOP,do_exit                         ;exit at memory function
3c1            b7 ac ;JPC ROTFC,do_addsub                       ;do the addsub for the next
number, if there is instruction for it
3c3            8a    ;SET MODEAS                                ;mark, that last operation was
add or sub
3c4            ef    ;CLR MENT + CLR OVFL + RET                 ;exit

;
;"C" Clear:     clear WR,DR,SR,TR and print. it does not clear RR,CR and RR.S2
;
3c5 fn_clear:  82    ;CLR MOP
3c6            49    ;CLR DR
```

```
3c7             d9    ;MOV WR,TR + CLR TR + CLR SR
3c8             4a    ;CLR WR
3c9             fc    ;PRN FPAR

;
;"CE" Clear:    clear WR, RR.S2, CR.S2
;
3ca fn_cleare: 4a    ;CLR WR
3cb             7f    ;CLR OVFL
3cc             f1    ;CLR MODE + CLR MENT + RET

;
;"Diamond" - subtotal: print the number or the subtotal
;
3cd fn_diamnd: 6c d5 ;JPC MODENN,dm_prn2                        ;jump in entry mode, print the
number, and close the entry mode
3cf             75 d3 ;JPC MODEMD,dm_prn1                        ;jump in mul/div mode, print the
number, and init
3d1             0b    ;MOV WR,SR                                 ;in add/sub mode, recall the
subtotal number from SR and clear SR
3d2             46    ;CLR SR
3d3 dm_prn1:   fc    ;PRN FPAR
3d4             ef    ;CLR MENT + CLR OVFL + RET
3d5 dm_prn2:   fd    ;PRN ROUND,FPAR
3d6             f1    ;CLR MODE + CLR MENT + RET

;entry address at digit, digit number, minus sign button
;                fuction code            parameter
;0..9            13                      0..9
;sign            13                      10
;digit point     13                      11
;00              6                       0
;000             12                      0

3d7 fn_digit:  d7    ;DIGIT                                     ;save digit into R13, place of
digit point (WR.S1) into R11
                                                                 ;at first entry: WR=0, CR.S2=8
3d8             a9 df ;JPC BIG_DIGIT,dig_dpsgn                   ;jump at digit point, minus sign

3da dig_numsh: 53    ;SHL WR                                    ;rotate the number into WR
3db             9a e3 ;JPC NBIG_WR,dig_chkdp                     ; jump, if there is now overflow
3dd             5f    ;SHR WR                                    ;at overflow, rotate back the
number (additional digits are lost)
3de             f3    ;CLR MODE + RET                            ;mark that new number is entered
since the last operation, and exit

3df dig_dpsgn: bc 5f ;JPC ODDPAR,dp_mark                        ;digit point button is pressed

3e1             e7    ;NEG WR                                    ;minus sign button is pressed
3e2             f3    ;CLR MODE + RET                            ;mark that new number is entered
since the last operation, and exit

3e3 dig_chkdp: 74 e8 ;JPC MENTDP,dig_incdp                      ;if digit point is already
entered, jump to adjust it
3e5             a7 ee ;JMP dig_nextd

3e7             00    ;(unimplemented, never used)

3e8 dig_incdp: ca    ;INC DPCNT                                 ;adjust the digit point place
with one digit more
3e9             ce ed ;JPC NBIG_DPCNT,dig_savdp
3eb             dd    ;DEC DPCNT                                 ;if already too much digit
entered after the digit point,
3ec             5f    ;SHR WR                                    ; ignore the new digit
3ed dig_savdp: c2    ;SET DP_WR                                 ;save the place of digit point
```

```
3ee dig_nextd: b7 da ;JPC ROTFC,dig_numsh                        ;function code contains, how
many '0's has to be entered yet
                                                                  ;implementation of button '00'
and '000' is here
3f0            f3    ;CLR MODE + RET                              ;mark that new number is entered
since the last operation, and exit

;
;Exchange function:     CR=WR, WR <- RR <- DR <- WR
;
3f1 fn_ex:     fd    ;PRN ROUND,FPAR
3f2            02    ;MOV CR,WR                                   ;CR=WR (WR is saved to CR)
3f3            0e    ;MOV WR,DR
3f4            03    ;MOV RR,WR                                   ;RR=DR
3f5            0c    ;MOV WR,CR
3f6            04    ;MOV DR,WR                                   ;DR=saved WR
3f7            0d    ;MOV WR,RR                                   ;WR=RR
3f8            f1    ;CLR MODE + CLR MENT + RET

;
;Clear memory:  recall (WR=MR), print and clear (R7=0)
;
3f9 fn_clrmem: 09    ;MOV WR,MR
3fa            fa    ;PRN FPAR,MEM
3fb            44    ;CLR MR
3fc            f1    ;CLR MODE + CLR MENT + RET

;
;Recall memory: recall (WR=MR) and print
;
3fd fn_rm:     09    ;MOV WR,MR
3fe            fa    ;PRN FPAR,MEM
3ff do_exit:   f1    ;CLR MODE + CLR MENT + RET

;----------------------------------------------------------------------------------------------------
------------------------------------
; Optional program for making the SQRT function
;----------------------------------------------------------------------------------------------------
------------------------------------

400 20 28           fim 0< $28          ;pseudo code entry address of the SQRT function

;
;Similar pseudo code interpreter implementation, like at $04b-05f, just uses the pseudo
instruction codes from address range $400-$4ff
;

402 11 06 jcnb_402: jcn TZ $406         ;wait for the inactive printer drum sector signal
404 50 b0           jms $0b0            ;keyboard handling
406 26 20 jcnf_406: fim 3< $20
408 28 10           fim 4< $10
40a 32              fin 1<              ;fetch pseudo instruction code into R2R3
40b f0              clb
40c 54 50           jms $450            ;execute the associated routine
40e 71 11 jcnb_40e: isz 1 $411          ;inc R0R1, pseudo code instruction pointer
410 60              inc 0
411 14 02 iszf_411: jcn AZ $402         ;jump back, if ACC returned by the pseudo instruction
was 0
413 f7              tcc
414 14 0e           jcn AZ $40e         ;if CY returned by the pseudo instruction was 0, R0R1 is
incremented again
416 30              fin 0<              ;if CY was set to 1, read the pseudo code jump address
417 44 02           jun $402            ;jump to continue the pseudo code from the modified
address

419     00 00 00 00 00 00 00            ;unused NOPs
```

```
420 00 00 00 00 00 00 00 00

;----------------------------------------------------------------------------------------------------
-----------------------------------
;Square root pseudo code implementation
;----------------------------------------------------------------------------------------------------
-----------------------------------

428 sq_start:  51    ;PRN FCODE                                       ;print number with function code
(9: SQRT)
429            a7    ;MOV CR,WR                                       ;save the number to the constant
register
42a            53    ;CLR RR                                          ;clear result register
42b            61 3e ;JPC ZERO_WR,sq_exit                             ;jump, if number is zero (the
result will be also zero)
42d            65    ;CLR DIGIT + GET DP_WR                           ;R10R11=place of digit point
42e sq_bshift: 63 44 ;JPC NBIG_WR,sq_lshift                           ;number is adjusted to the
leftmost position
430            9c    ;SHR WR                                          ;one digit overshift is
corrected back
431            5b    ;MOV DR,WR                                       ;remainder (DR) is initialized
to the shifted number
432            55    ;CLR WR                                          ;initial subtrahend (WR) is
cleared
433            6a 36 ;SET LPCSQRT + SET DPCNTSQRT + JPC EVENDP,sq_loopns        ;R15=13, sqrt
digit point calculation
                                                                      ;jump if original digit point
position was even
435 sq_loopsh: 58    ;SHL DR                                          ;multiplication by 10 of the
remaining part
                                                                      ;(and possible additional shift
if it is needed)
436 sq_loopns: 7a    ;INC WR_POS                                      ;increment the subtrahend (WR
from position in R15) by 1
437            5d 41 ;SUB DR,WR + JPC NNEG,sq_rptinc + INC DIGIT;remainder is decremented by the
subtrahend (DR=DR-WR)
                                                                      ;and jump, if the result is not
negative
                                                                      ;digit counter (R13) is
incremented too
439            5f    ;ADD DR,WR                                       ;add the subtrahend to get back
the last non negative value
43a            85    ;DEC WR_POS                                      ;decrement the subtrahend by one
(prepare it for the next round)
43b            57    ;SHL RR                                          ;shift the new digit into the
number, R13 is cleared too
43c            98 35 ;JPC NZERO_LPCSQRT,sq_loopsh + DEC LPCSQRT ;decrement R15, and jump, except
when R15 becomes 0
                                                                      ;(next round calculates with one
more digit)
43e sq_exit:   a9    ;MOV DR,WR (MOV WR,CR ???)                       ;??? subtrahend is saved
(originally it may be WR=CR)
43f            5b    ;MOV DR,WR                                       ;??? duplicated, but not
disturbing code
440            9f    ;CLR MOP + RET_BPC                               ;return back to basic pseudo
code interpreter to address $40
441 sq_rptinc: 7a    ;INC WR_POS                                      ;increment the subtrahend by 1
(WR from position in R15)
442            96 36 ;JMP sq_loopns                                   ;jump back

444 sq_lshift: 59    ;SHL WR                                          ;rotate number into left
position
445            93 2e ;INC DPCNT + JMP sq_bshift                       ;increment R10R11, and jump back

;----------------------------------------------------------------------------------------------------
-----------------------------------
```

```
447                         00                ;unused NOPs
448 00 00 00 00 00 00 00 00

450 33    jmsf_450: jin 1<              ;jump to the pseudo instruction code associated routine

451 41 fe vmbc_451: jun $1fe            ;PRN FCODE
453 41 48 vmbc_453: jun $148            ;CLR RR
455 41 4a vmbc_455: jun $14a            ;CLR WR
457 68    vmbc_457: inc 8               ;SHL RR
458 68    vmbc_458: inc 8               ;SHL DR
459 41 53 vmbc_459: jun $153            ;SHL WR
45b 41 04 vmbc_45b: jun $104            ;MOV DR,WR
45d 41 34 vmbc_45d: jun $134            ;SUB DR,WR + JPC NNEG + INC DIGIT
45f 41 21 vmbc_45f: jun $121            ;ADD DR,WR
461 41 a2 vmbc_461: jun $1a2            ;JPC ZERO_WR
463 41 9a vmbc_463: jun $19a            ;JPC NBIG_WR

;QPC_65:         CLR DIGIT + GET DP_WR

465 bd    vmbc_465: xch 13              ;clear digit (R13=0)
466 29              src 4<
467 ed              rd1
468 bb              xch 11              ;R11=WR.S1, get the digit point place of WR
469 c0              bbl 0

;QPC_6A:         SET LPCSQRT + SET DPCNTSQRT + JPC EVENDP
;                R15=13, R10R11=(R10R11/2+6+((R10R11 mod 2))), jump, if original R10R11 was even

46a 2e 6d vmbc_46a: fim 7< $6d          ;R14=6, R15=13
46c ab              ld 11
46d b7              xch 7               ;R7=R11 (save original R11 into R7)
46e ba              xch 10              ;ACC=R10  (R10=0 [previous R7])
46f f6              rar                 ;CY=R10.bit0
470 ab              ld 11
471 f6              rar                 ;ACC=8*(R10.bit0)+(R11 div 2),  CY=(R11 mod 2)
472 8e              add 14              ;ACC=8*(R10.bit0)+(R11 div 2)+(R11 mod 2)+6, CY=overflow
473 bb              xch 11              ;store it to R11
474 f7              tcc                 ;ACC=overflow
475 ba              xch 10              ;R10=0 or 1
476 b7              xch 7               ;ACC=original R11
477 f6              rar                 ;CY=(R11 mod 2), rotate bit 0 into CY
478 f3              cmc                 ;CY=1-(R11 mod 2), negate the pseudo jump condition
479 c1              bbl 1               ;prepare pseudo code jump

;QPC_7A:         INC WR_POS      increment WR from position in R15

47a af    vmbc_47a: ld 15
47b b9              xch 9               ;R9=R15
47c fa              stc
47d d0              ldm 0               ;clear ACC
47e 29              src 4<
47f eb              adm
480 fb              daa                 ;add carry to number digit by digit
481 e0              wrm
482 79 7d           isz 9 $47d          ;loop back for the next digits
484 c0              bbl 0

;QPC_85:         DEC WR_POS      Decrement WR from position in R15
;
;inside the loop when R7 is subtracted from ACC and CY is complemented:
;
;                CY=0                        CY=1
;-------------------------------------------------------
;ACC 0           ACC=0, CY=0                 ACC=15->9, CY=1
;ACC 1..9        ACC=ACC, CY=0               ACC=ACC-1, CY=0
```

```
485 af    vmbc_485: ld 15
486 b9              xch 9               ;R9=R15
487 f3              cmc                 ;at first: set CY=1, later complement the borrow bit
488 29              src 4<
489 e9              rdm                 ;read next digit from WR
48a 97              sub 7               ;subtract R7 (=0) from it, ACC=ACC+15+(1-CY)
48b 12 8e           jcn C1 $48e         ;jump, if there is no borrow
48d d9              ldm 9               ;set the number to 9 (BCD adjust)
48e e0              wrm                 ;write back the result
48f 79 87           isz 9 $487          ;loop back for the next digits
491 f0              clb
492 c0              bbl 0

;QPC_93:        INC DPCNT + JMP         Increment digit point counter (R10R11) and unconditional jump
;QPC_96:        unconditional jump

493 7b 96 vmbc_493: isz 11 $496         ;inc R11, and skip if result is nonzero
495 6a              inc 10              ;inc R10
496 fa    vmbc_496: stc                 ;set CY=1, the pseudo jump condition
497 c1              bbl 1               ;prepare pseudo code jump

;QPC_98:        JPC NZERO_LPCSQRT + DEC LPCSQRT         decrement R15, and jump, except when R15 was 0

498 af    vmbc_498: ld 15               ;decrement R15, sqrt loop counter
499 f8              dac
49a bf              xch 15              ;the pseudo jump condition is set, if R15 was nonzero
49b c1              bbl 1               ;prepare pseudo code jump

;QPC_9C:        SHR WR          Right shift of working register
49c 41 5f vmbc_49c: jun $15f            ;one digit right shift of WR with R13 (0 is shifted from left)

49e 00              nop

;QPC_9F:        CLR MOP + RET_BPC       Clear divide/multiply operation and return back to basic pseudo code interpreter

49f 27    vmbc_49f: src 3<              ;clear DR.S2
4a0 e6              wr2
4a1 20 40           fim 0< $40          ;entry address is $40
4a3 26 00           fim 3< $00
4a5 40 4b           jun $04b            ;jump back to basic pseudo code interpreter

;QPC_A7:        MOV CR,WR       Move working register into constant register (CR=WR)
4a7 41 02 vmbc_4a7: jun $102            ;CR=WR

;QPC_A9:        MOV DR,WR (or MOV WR,CR)
;       Move working register into dividend/multiplicand register (DR=WR), but it is very probable that this would be
;       move constant register into working register (WR=CR)

4a9 41 04 vmbc_4a9: jun $104            ;Maybe it is "jun $10c"
                                        ;(the difference is only one bit in the code - was the source ROM damaged?)

4ab          00 00 00 00 00             ;Unused NOPs
4b0 00 00 00 00 00 00 00 00
4b8 00 00 00 00 00 00 00 00
4c0 00 00 00 00 00 00 00 00
4c8 00 00 00 00 00 00 00 00
4d0 00 00 00 00 00 00 00 00
4d8 00 00 00 00 00 00 00 00
4e0 00 00 00 00 00 00 00 00
```

```
4e8 00 00 00 00 00 00 00 00
4f0 00 00 00 00 00 00 00 00
4f8 00 00 00 00 00 00 00 00

;-----------------------------------------------------------------------------------------------------------------------------
```