Table of Contents

[PC-1500 System RAM 4](#_Toc91513330)

[Display Buffer (7600-774F) 4](#_Toc91513331)

[Fixed String variables (7650-77AF) 5](#_Toc91513332)

[System RAM Map (7800-79BF) 5](#_Toc91513333)

[Fixed Variables (78C0-79CF) 8](#_Toc91513334)

[System RAM Map, Options (79D0-79FF) 8](#_Toc91513335)

[Basic Registers (7A00-7A37) 9](#_Toc91513336)

[The Basic Stack 10](#_Toc91513337)

[System RAM Map (7B00-7B0F) 11](#_Toc91513338)

[String buffer 12](#_Toc91513339)

[Output Buffer 12](#_Toc91513340)

[Input Buffer 13](#_Toc91513341)

[Usable ROM Subroutines 13](#_Toc91513342)

[Clearing and Moving 13](#_Toc91513343)

[Blocks of Memory 13](#_Toc91513344)

[Clearing of Buffers 14](#_Toc91513345)

[Data Transfers to Buffers 14](#_Toc91513346)

[Floating-Point Arithmetic 14](#_Toc91513347)

[Preloading of XH and YH 15](#_Toc91513348)

[Data transfer between R0 and fixed variables 15](#_Toc91513349)

[Clearing of registers 15](#_Toc91513350)

[Data transfer between registers 15](#_Toc91513351)

[Data transfer between R0 and Basic Stack 15](#_Toc91513352)

[Arithmetic Operations 16](#_Toc91513353)

[Comparisons 16](#_Toc91513354)

[Constants (From ROM table) into R2 17](#_Toc91513355)

[Separation of integer and fraction 17](#_Toc91513356)

[Removal of sign 17](#_Toc91513357)

[Normalization of R0 17](#_Toc91513358)

[Binary to decimal conversion 17](#_Toc91513359)

[Decimal to binary conversion 18](#_Toc91513360)

[Binary Routines 19](#_Toc91513361)

[Storage and recall of two-byte data: 19](#_Toc91513362)

[Operating in binary 19](#_Toc91513363)

[Binary format in R0 19](#_Toc91513364)

[Character String Routines 20](#_Toc91513365)

[String Pointer 20](#_Toc91513366)

[Equivalents of Basic functions: 20](#_Toc91513367)

[String comparisons 20](#_Toc91513368)

[Variables used by Basic 21](#_Toc91513369)

[Locating a named variable 21](#_Toc91513370)

[Loading R0 with variable contents 21](#_Toc91513371)

[Clearing variables 22](#_Toc91513372)

[TIME and RANDOM Number 22](#_Toc91513373)

[TIME (Format used by Basic) 22](#_Toc91513374)

[TIME (Alternate format) 22](#_Toc91513375)

[Random Number 22](#_Toc91513376)

[Reading codes from Basic 23](#_Toc91513377)

[Evaluation of expressions in Basic 23](#_Toc91513378)

[Conversion of hex characters to byte 23](#_Toc91513379)

[Search for Basic program line 23](#_Toc91513380)

[Keyboard 24](#_Toc91513381)

[Poll of keyboard 24](#_Toc91513382)

[Wait for input form keyboard 24](#_Toc91513383)

[Detection of BREAK key 24](#_Toc91513384)

[Display 25](#_Toc91513385)

[Cursor Pointer 25](#_Toc91513386)

[To clear display 25](#_Toc91513387)

[To display data in R0 (or string pointed to by R0) 25](#_Toc91513388)

[To display numeric contents of R0, formatted 25](#_Toc91513389)

[To display a string pointed to be R0 25](#_Toc91513390)

[To display contents of output buffer 25](#_Toc91513391)

[To display a sequence of characters 26](#_Toc91513392)

[To display a program line 26](#_Toc91513393)

[To display the contents of input buffer with tokens expanded 26](#_Toc91513394)

[To display a single character 27](#_Toc91513395)

[Graphic Display 27](#_Toc91513396)

[Timed Delay 27](#_Toc91513397)

[BEEP 27](#_Toc91513398)

[Power Down 27](#_Toc91513399)

[Termination at ‘READY’ 28](#_Toc91513400)

[Terminate with display 28](#_Toc91513401)

[Terminate with prompt mark 28](#_Toc91513402)

[Terminate with error message 28](#_Toc91513403)

[Printer Routines 29](#_Toc91513404)

[Motor Off 29](#_Toc91513405)

[Printing in text mode 29](#_Toc91513406)

[Plotting (GRAPH mode) 30](#_Toc91513407)

[Printing in GRAPH mode 30](#_Toc91513408)

[Printing program lines (Text mode) 31](#_Toc91513409)

[Miscellaneous (Mechanical) 31](#_Toc91513410)

[Cassette Routines 32](#_Toc91513411)

[To save a file on cassette 32](#_Toc91513412)

[To load a file form cassette 32](#_Toc91513413)

[Control of relays 33](#_Toc91513414)

[Record, Load, or verify block of data 33](#_Toc91513415)

[Record or load a single byte 33](#_Toc91513416)

This is a compilation of a series of articles by Norlin Rober which appeared in Pocket Computer Newsletter, Issues 31-36 in 1984. I hope having them in one document will make it easier for folks, who are still interested in vintage pocket computers, to reference. Jeff Birt, December 2021

# PC-1500 System RAM

Addresses 7600-7C00 contain the Display Buffer, Input Buffer, fixed variables, and various other RAM required by the system.

Because of incomplete decoding the memory located 7600-77FF may also be addresses as 7000-71FF, 7200-73FF, or 7400-75FF.

In the PC-1500 the memory located 7800-7BFF is also addressable as 7C00-7FFF. In some versions of the PC-1500 this will be read only.

In the PC-1500A the area 7C00-7FFF contains RAM intended for storage of machine language programs.

## Display Buffer (7600-774F)

Each dot on the LCD is turned on by the presence of a ‘1’ bit in a corresponding location in the Display Buffer. The upper four dots (U) in a column of the LCD are determined by one nibble and the lower three dots (L) by another, *as tabulated below:*

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Memory Address | Columns Controlled Low nibble Hi Nibble | | Memory Address | Columns Controlled Low nibble Hi Nibble | |
| 7600 | 0 U | 78 U | 7700 | 39 U | 117 U |
| 7601 | 0 L | 78 L | 7701 | 39 L | 117 L |
| 7602 | 1 U | 79 U | 7702 | 40 U | 118 U |
| 7603 | 1 L | 79 L | 7703 | 40 L | 118 L |
| 7604 | 2 U | 80 U | 7704 | 41 U | 119 U |
| 7605 | 2 L | 80 L | 7705 | 41 L | 119 L |
| . . . | . . . | . . . | . . . | . . . | . . . |
| 764C | 38 U | 116 U | 774C | 77 U | 155 U |
| 764D | 38 L | 116 L | 774D | 77 L | 155 L |

*Display annunciators are set by bits as follows:*

|  |  |
| --- | --- |
| 764E | Misc: 01=BUSY, 02=SHIFT, 04=Katakana characters, 08=SMALL, 10=III, 20=II, 40=I, 80=DEF |
| 764F | Modes: 01=DE, 02=G, 04=RAD, 10=RESERVE, 20=PRO, 40=RUN |
| 774E | Unused by PC-1500 or CE-150 |
| 774F | Unused by PC-1500 or CE-150 |

## Fixed String variables (7650-77AF)

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 7650-5F | E$ | 76B0-BF | K$ | 7750-5F | P$ | 77B0-5F | V$ |
| 7660-6F | F$ | 76C0-CF | L$ | 7760-6F | Q$ | 77C0-CF | W$ |
| 7670-7F | G$ | 76D0-DF | M$ | 7770-7F | R$ | 77D0-DF | X$ |
| 7680-8F | H$ | 76E0-EF | N$ | 7780-8F | S$ | 77E0-EF | Y$ |
| 7690-9F | I$ | 76F0-FF | O$ | 7790-9F | T$ | 77F0-FF | Z$ |
| 76A0-AF | J$ |  |  | 77A0-AF | U$ |  |  |

## System RAM Map (7800-79BF)

|  |  |
| --- | --- |
| 7800-4F | Comprise the stack used by the CPU. Storage into this stack begins at 784F, continuing in the reverse direction through memory. |
| 7850 | Used by CE-158 |
| 7851 | Used by CE-158 |
| 7852 | Used by CE-158 |
| 7853 | Used by CE-158 |
| 7854 | Used by CE-158 |
| 7855 | Used by CE-158 |
| 7856 | Used by CE-158 |
| 7857 | Used by CE-158 |
| 7858 | Used by CE-158 |
| 7859 | Used by CE-158 |
| 785A | Used by CE-158 |
| 785B | High byte, address of external character input routine |
| 785C | Low byte, external character input routine |
| 785D | If 80=Katakana displayed, 00=displayed and printed, FF=neither |
| 785E | High byte, location of the table of Katakana character codes |
| 785F | Unused |
| 7860 | High byte, START OF ROM in module, FF is no module present |
| 7861 | High byte, START OF BASIC program in ROM module, FF is no module. |
| 7862 | Low byte, START OF BASIC program in module, FF id no module |
| 7863 | High byte, START OF RAM |
| 7864 | High byte, TOP OF RAM |
| 7865 | High byte, START OF BASIC program in RAM |
| 7866 | Low byte, START OF BASIC program in RAM |
| 7867 | High byte, END OF BASIC program (stop byte address) |
| 7868 | Low byte, END OF BASIC program |
| 7869 | High byte, START OF EDIT |
| 786A | Low byte, START OF EDIT |
| 786B | Flags: 01=BEEP OFF, 80=RMT ON |
| 786C | Used by ROM to temporarily save contents |
| 786D | of 787C-7F while Display Buffer |
| 786E | is temporarily saved into the |
| 786F | string and Output Buffers. |
| 7870 | Unused |
| 7871 | WAIT setting: 01=WAIT, 02=WAIT nnnn, 03-WAIT 0 |
| 7872 | High byte, WAIT time counter |
| 7873 | Low byte, WAIT time counter |
| 7874 | Flags: 01=Cursor enabled, 80=display currently saved 7B10-7BAB |
| 7875 | CURSOR POINTER (current display column number) |
| 7876 | Character position number is display, with INPUT statement |
| 7877 | Compliment of number of display positions left for INPUT prompt |
| 7879 | Cassette Parameter: 1=MERGE, 2=CHAIN, 4=Data file, 8=ML, 10=EMT 1, 40=CLOAD?, 80=load operation |
| 787A | Unused |
| 787B | Position of blink character in display, plus 8 |
| 787C | Flags: 01=blink cursor enabled, 80=a character is now blinked |
| 787D | Code of character blinked |
| 787E | High byte, location (in Display Buffer) of Blink Cursor |
| 787F | Low byte, location of Blink Cursor |
| 7880 | Display Parameter: determines the display at READY depending on bits set. 20=display contents of R0, otherwise contents of Input Buffer. 10=display as BASIC line. 04=include colon after line number. 40=cursor activated in display. 80=ERROR message displayed. 08=RESERVE key phrase displayed. 01=RESERVE template displayed, with previous display saved in 7B10-7BAB. |
| 7881 | Parsing Parameter: specifies type of instruction that will conform to the rules of syntax for evaluating BASIC expression. |
| 7882 | Sub-pointer, used with GOSUB and FOR stacks. Also, 01 indicates a USING statement that is not part of a PRINT or PAUSE statement. |
| 7883 | High byte, LET pointer. Also, 00 indicates no data passed w/ Call |
| 7884 | Low byte, LET pointer. Also, used to store code of RESERVE key |
| 7885 | With LET pointer: length if string, 88 if numeric variable |
| 7886 | High byte, INPUT pointer (starting address of variable) |
| 7887 | Low byte, INPUT pointer |
| 7888 | With INPUT pointer: length if string, 88 if numeric variable |
| 7889 | Flag: 01=variable subscript(s) being parsed |
| 788A | Program Halt Parameter: 10=waiting for input, 20=halt for PRINT, 40=INPUT statement pending, 80=break |
| 788B | Low byte, Input Buffer pointer |
| 788C | Number of function input arguments (used with parsing routine) |
| 788D | TRACE: 00=OFF, other=ON (Contents of 79D1 stored here by TRON) |
| 788E | TRACE Parameter: 00=ending execution of previous line, 01=starting execution of new line, 02=expression in Input Buffer is to be evaluated, 04=expression has been evaluated (immediate mode) |
| 788F | Low byte, Output Buffer pointer |
| 7890 | Low byte, FOR stack pointer |
| 7891 | Low byte, GOSUB stack pointer |
| 7892 | Low byte, BASIC DATA STACK pointer, used by parsing routine |
| 7893 | Low byte, BASIC PENDING OP STACK pointer, used by parsing routine |
| 7894 | Low byte, String Buffer Pointer |
| 7895 | USING editing character: 10=comma separation, 20=forced sign, 40=asterisk fill, 80=scientific. 01 used to check syntax in interpretation of USING statement. |
| 7896 | USING number of characters, including sign, before decimal point |
| 7897 | USING number of characters in string: 00=unspecified |
| 7898 | USING number of characters including and following decimal point |
| 7899 | High byte, START OF VARIABLES in main memory |
| 789A | Low byte, START OF VARIABLES in main memory |
| 789B | Error code |
| 789C | High byte, CURRENT line number: 00 if no program in progress |
| 789D | Low byte, CURRENT line number: 00 if no program in progress |
| 789E | High byte, beginning address of CURRENT program |
| 789F | Low byte, beginning address of CURRENT program |
| 78A0 | High byte, PREVIOUS address |
| 78A1 | Low byte, PREVIOUS address |
| 78A2 | High byte, PREVIOUS line number |
| 78A3 | Low byte, PREVIOUS line number |
| 78A4 | High byte, beginning of program containing PREVIOUS line |
| 78A5 | Low byte, beginning of program containing PREVIOUS line |
| 78A6 | High byte, SEARCH address |
| 78A7 | Low byte, SEARCH address |
| 78A8 | High byte, SEARCH line number |
| 78A9 | Low byte, SEARCH line number |
| 78AA | High byte, beginning of program containing SEARCH line |
| 78AB | Low byte, beginning of program containing SEARCH line |
| 78AC | High byte, BREAK address |
| 78AD | Low byte, BREAK address |
| 78AE | High byte, BREAK line number |
| 78AF | Low byte, BREAK line number |
| 78B0 | High byte, beginning of program containing BREAK line |
| 78B1 | Low byte, beginning of program containing BREAK line |
| 78B2 | High byte, ERROR address |
| 78B3 | Low byte, ERROR address |
| 78B4 | High byte, ERROR line number |
| 78B5 | Low byte, ERROR line number |
| 78B6 | High byte, beginning of program containing ERROR line |
| 78B7 | Low byte, beginning of program containing ERROR line |
| 78B8 | High byte, ON ERROR address |
| 78B9 | Low byte, ON ERROR address |
| 78BA | High byte, ON ERROR line number |
| 78BB | Low byte, ON ERROR line number |
| 78BC | High byte, beginning of program containing ON ERROR line |
| 78BD | Low byte, beginning of program containing ON ERROR line |
| 78BE | High byte, DATA pointer: 80 indicates no DATA line yet located |
| 78BF | Low byte, DATA pointer |

## Fixed Variables (78C0-79CF)

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 78C0-CF | A$ | 7920-27 | E | 7958-5F | L | 7998-9F | T |
| 78D0-DF | B$ | 7928-2F | F | 7960-67 | M | 79A0-AF | U |
| 78E0-EF | C$ | 7930-3F | G | 7968-6F | N | 79A8-AF | V |
| 78F0-FF | D$ | 7938-2F | H | 7970-77 | O | 79B0-BF | W |
| 7900-07 | A | 7940-47 | I | 7978-7F | P | 79B8-BF | X |
| 7908-0F | B | 7948-4F | J | 7980-87 | Q | 79C0-CF | Y |
| 7910-17 | C | 7950-57 | K | 7988-8F | R | 79C8-CF | Z |
| 7918-1F | D |  |  | 7990-97 | S |  |  |

## System RAM Map, Options (79D0-79FF)

|  |  |
| --- | --- |
| 79D0 | ROM Bank: 00=ROM 1, 01=ROM 2 |
| 79D1 | OPN device code: 60=LCD, 5C=CMT, 58=MGP. C4=LPRT, C0=COM |
| 79D2 | Unused |
| 79D3 | If 55, bypass setting of modulation clock frequency, serial output |
| 79D4 | If 55, bypass keyboard scan, obtain input from external device |
| 79D5 | Unused |
| 79D6 | Unused |
| 79D7 | Unused |
| 79D8 | Unused |
| 79D9 | Used (by option) |
| 79DA | If 55, interrupt routine address in option obtained from 79DB-DC |
| 79DB | High byte, interrupt routine address in option |
| 79DC | Low byte, interrupt routine address in option |
| 79DD | Unused |
| 79DE | Unused |
| 79DF | Unused |
| 79E0 | High byte of X coordinate, in signed binary (GRAPH mode). Also used for high byte address of first line to be LLISTed |
| 79E1 | Low byte of X coordinate, in signed binary (GRAPH mode). Also used for high byte address of first line to be LLISTed |
| 79E2 | High byte of Y coordinate, in signed binary (GRAPH mode). Also used for high byte address of first line to be LLISTed |
| 79E3 | Low byte of Y coordinate, in signed binary (GRAPH mode). Also used for high byte address of first line to be LLISTed |
| 79E4 | High byte, paper reverse feed count (counts from 0001 to 01FF) |
| 79E5 | Low byte, paper reverse feed count |
| 79E6 | Location of Pen (00 to D8) |
| 79E7 | Low byte, extension of X-coordinate beyond available space |
| 79E8 | High byte, extension of X-coordinate beyond available space |
| 79E9 | Pen Parameter: specifies whether Pen is to be raised or lowered |
| 79EA | LINE TYPE (0 to 9), GRAPH mode |
| 79EB | Dotted line counter |
| 79EC | Current Pen position: 00=up, 01=down |
| 79ED | X-motor hold counter |
| 79EE | Motor Phase: stored in Port C |
| 79EF | Y-Motor hold counter |
| 79F0 | Print Mode: 00=TEXT, FF=GRAPH |
| 79F1 | Printer disable: 0F=Pen change mode, FF=low battery / head blocked |
| 79F2 | ROTATE setting (0 to 3) |
| 79F3 | COLOR setting (0 to 3) |
| 79F4 | CSIZE setting (0 to 3) |
| 79F5 | LPRINT Parameter: 00=single item, 04=comma LPRINT, 08=semi-colon LPRINT, 01=last item reached. Also used with LLIST for maximum permissible line length. |
| 79F6 | With LINE, used as direction parameter. With LLIST, to determine line feed. With COLOR, to save Pen location. |
| 79F7 | Type of data LRPINTed: 00=numeric, FF=character string |
| 79F8 | Temporary storage of Pen location during paper feed |
| 79F9 | Flag: flag indicating power up or interrupt in progress |
| 79FA | Used by CE-158 |
| 79FB | Used by CE-158 |
| 79FC | Used by CE-158 |
| 79FD | Used by CE-158 |
| 79FE | Used by CE-158 |
| 79FF | LOCK mode: 00=LOCK, FF=UNLOCK |

## Basic Registers (7A00-7A37)

*Seven registers used in Basic computation are as follows:*

|  |  |  |  |
| --- | --- | --- | --- |
| Address Range | # | Label | Description |
| 7A00-7A07 | R0 | AR-X | Floating-Point accumulator |
| 7A08-7A0F | R1 | AR-Z | Scratch register |
| 7A10-7A17 | R2 | AR-Y | Second operand |
| 7A18-7A1F | R3 | AR-U | Scratch register |
| 7A20-7A27 | R4 | AR-V | Scratch register |
| 7A28-7A2F | R5 | AR-W | Scratch register |
| 7A30-7A37 | R6 | AR-S | Temporary storage register |

Each of these registers contain a two-byte signed-binary number, a floating-point decimal number, or a pointer to a character string.

*The eight bytes in a register are utilized as follows:*

|  |  |  |  |
| --- | --- | --- | --- |
| Byte | Signed Binary | Floating Point | String Pointer |
| 0 | Unused | Exponent (binary) | Unused |
| 1 | Unused | Sign: 80 specifies ‘-‘ | Unused |
| 2 | Unused | 1st & 2nd digits | Unused |
| 3 | Unused | 3rd & 4th digits | Unused |
| 4 | B2 (to identify) | 5th & 6th digits | D0 (to identify) |
| 5 | 1st byte | 7th & 8th digits | High byte, address |
| 6 | 2nd byte | 9th & 10th digits | Low byte, address |
| 7 | Unused | 11th & 12th digits | Number of characters |

Mantissa of calculated floating-point numbers are rounded to ten digits prior to display, printing or storage. A string pointer formed by execution of CHR$ uses C1 instead of D0 as byte 4.

*The area 7A00-7A37 is also used for other purposes, as follows:*

|  |  |
| --- | --- |
| 7A20 | 00=power-down by 'OFF' key, 01=timed power-down, 02=other type power down, 04=memory allocation pointers have changed |
| 7A21 | Set to other than zero by option with low battery power, indicating need for display of 'CHECK' message. |
| 7A22 | Set by option to indicate need for 'CHECK' message |

A power down resulting from 7 minutes of non-use is identified by storage of A0,A1,…,AF into addresses 7A10-7AFF, the stack pointer is saved in 7A30-31. The power-down initiated by use of the OFF-key stores 50,51,…,5F into 7A10-7AFF.

*The following are used in calculations of transcendental functions:*

|  |  |
| --- | --- |
| 7A18 | Flag: 00=SIN, COS or ASN; 01=LOG, LN, or EXP; 20=ACS; 40=TAN or ATN; 80=result must be subtracted from 90 (inverse trig) |
| 7A20 | Flag: 00=SIN; 01=COS |

The PRINT and PAUSE routines use the area 7A10-7A34 to form a string of characters that represent numeric data formatted for display. The same area is used to form an ERROR or BREAK message. The area beginning 7A08 is used as a scratch area in the interpretation of USING statements and in execution of certain printer routines. In some of the printing routines. The contents of CPU register Y are saved in 7A26-7A27.

## The Basic Stack

The Basic Stack, located 7A38-7AFF, includes a FOR stack, a GOSUB stack, and an arithmetic stack. The FOR-loop stack begins at 7A38. Up to 16 loops may be pending at a given time if the Basic stack is not required for other purposes.   
  
*Each FOR register contains 12 bytes, used as follows:*

|  |  |
| --- | --- |
| 0-1 | Memory address of control variable |
| 2-3 | Test value, in signed binary |
| 4-5 | Step size, in signed binary |
| 6-7 | Address of next statement following FOR, plus 80 if not the first statement in a line. |
| 8-9 | Number of the line containing the statement following FOR |
| A-B | Address of beginning of program in progress |

GOSUB return addresses are stacked beginning at 7AFF, proceeding in reverse direction through memory. Up to 33 return addresses may be pending if the Basic stack is not required for other purposes.   
  
*Each GOSUB register contains 6 bytes, as follows:*

|  |  |
| --- | --- |
| 0-1 | Address of statement following GOSUB, plus 80 if not the first statement in a line. |
| 2-3 | Number of the line containing the statement following GOSUB |
| 4-5 | Address of beginning of program in progress |

The space between the FOR and GOSUB stacks compromises the arithmetic stack which is used by the parsing routine in evaluating expressions. Pending numeric data, or string pointer, will be stacked, in 8-byte blocks, starting at 7A3C, or at the location for bytes after the last address used for storing a FOR register. Codes representing pending operations are stacked in reverse direction through memory, starting at 7AFF, or at the location just preceding the stacked GOSUB registers.

Each pending operation code contains two bytes, of which the first indicated the priority of the operation. Those representing functions consist of the token codes. The tokens for SQR and PI represent the corresponding symbols. The corresponding symbols.   
  
*The other pending operation codes follow:*

|  |  |  |  |
| --- | --- | --- | --- |
| 84 5E | ^ | 70 51 | OR |
| 83 21 | Sign change | 70 50 | AND |
| 82 2F | / | 60 2C | Comma separating arguments |
| 82 2A | \* |  |  |
| 81 2D | - | 5A xx | Variable name |
| 81 2B | + | 59 xx | Variable name |
| 80 06 | >= |  |  |
| 80 05 | <= | 41 xx | Variable name |
| 80 04 | = |  |  |
| 80 02 | > | 40 80 | @< |
| 80 01 | < | 20 28 | Left parenthesis |
| 80 00 | <> |  |  |

## System RAM Map (7B00-7B0F)

|  |  |
| --- | --- |
| 7B00-7B07 | Contain the current value of the random number |
| 7B08 | Unused |
| 7B09 | Counter for time that a key is held down |
| 7B0A | High byte, timed power down counter |
| 7B0B | Second byte, timed power down counter |
| 7B0C | Low byte, timed power down counter |
| 7B0D | Cursor blink counter. Counts from 80 to FF |
| 7B0E | Cursor Control Parameter: 80=indicates down arrow key required to continue program execution. Other bits control cursor repeat. |
| 7B0F | Key matrix code for depressed key. |

## String buffer

The 80-byte String Buffer, located 7B10-7B5F, contains the codes for character strings formed by the parsing of string expressions.

The string buffer and output buffer together form a temporary storage area into which the previous contents of the Display Buffer are saved during display of a program line, at an execution halt, or during the display of a BREAK message or RESERVE template.

Certain printing routines use portions of the String Buffer as a scratch area. Address 7B1F is used as a flag containing 00 for LLIST and FF for auto print of the Input Buffer contents in immediate mode.

## Output Buffer

The 80-byte Output Buffer is located 7B60-7BAF

The codes for the characters to be displayed in response to a PRINT, PAUSE, or immediate calculation, are stored in the required format, in address 7B60-7B79. Codes for numeric output are first placed into the 7A10-34 area, then transferred here to construct the display. Codes for string output are transferred from the string buffer.

With LPRINT the characters representing numeric data to be printed are stored, in the required format, in the area 7B80-7B9E. Another portion of the Output Buffer, ending 7B7F, serves as a scratch area used to form printed characters.   
  
*Other locations are used as follows:*

|  |  |
| --- | --- |
| 7BA9 | A line feed, following printing, is specified by 04 |
| 7BAA | USING editing character: 10=comma separation, 20=forced sign, 40=asterisk fill, 80=scientific |
| 7BAB | USING number of characters preceding decimal point |
| 7BAC | USING number of characters in string: 00 if unspecified |
| 7BAD | USING number of characters including and following decimal point |
| 7BAE | High byte, saved value of register Y |
| 7BAF | Low byte, saved value of register Y |

Graphic printing routines use the area 7B92-7BAD to store coordinates of points specified by LINE or RLIN statements as well as of the present location.

*The following addresses are also used:*

|  |  |
| --- | --- |
| 7B83 | Flag: 00=Line, FF=Box |
| 7B84 | Count of number of line segments to be drawn |
| 7B85 | Flag: 00=Line, nonzero=RLINE |

Prior to recording or locating a cassette file a header is formed in the Output Buffer, *located as follows:*

|  |  |
| --- | --- |
| 7B60-67 | Lead-in: consists of the bytes 10,11,…,19 |
| 7B68 | File type: 00=ML, 01=Basic, 02=RESERVE, 04=data |
| 7B69-78 | File name: null bytes if unspecified |
| 7B79-81 | Unused |
| 7B82-83 | Beginning memory addresses of file, not used for data file |
| 7B84-85 | Number of bytes in file, less 1. Zero for data files |
| 7B86-87 | Starting address for automatic execution, ML files only. If not specified, then FFFF |

During a load operation header information read form the recorded file is stored into the Output Buffer *as follows:*

|  |  |
| --- | --- |
| 7B91-A0 | File name |
| 7BA1-A9 | Unused |
| 7BAA-AB | Beginning memory address. Used when CLOAD M statement does not specify an address. |
| 7BAC-AD | Number of bytes in file, less 1. Zero for data file. |
| 7BAE-AF | Starting address for automatic execution of ML program |

## Input Buffer

The 80-byte Input Buffer, located 7BB0-7BFF, holds the codes for characters as they are entered from the keyboard. It is also used for storing codes that form input prompts, displayed program lines, expressions to be evaluated in immediate mode, and displayed RESERVE key phrases.

# Usable ROM Subroutines

The list which follows contains information related to a number of subroutines in ROM that are usable in user machine language programs.

A subroutine that is addressable as a Vector Call will be listed as ‘CALL nn’. The address of the subroutine will follow in parentheses.

## Clearing and Moving

### Blocks of Memory

Call BA (F763) clears the contents of memory from address X to X+UL. Register Y is not affected.

Subroutine D3C5 clears memory from address X to X+U. Register Y is not affected.

Subroutine D3C7 fills memory from address X to X+U with the byte contained in register A. Register Y is not affected.

### Clearing of Buffers

Call F2 (EE71) Clears the Display Buffer (7600-764D and 7700-774D). Register Y is not affected.

Subroutine D02B fills the Input Buffer with 0D codes. Both Register Y and the Input Buffer Pointer (788B) are set to point to 7BB0, the beginning address of the Input Buffer.

### Data Transfers to Buffers

Call 94 (EC5C) copies the contents of addresses X to X+UL+1 into the Output Buffer, starting at the position specified by the Output Buffer Pointer (788F). If insufficient room is available flag C will be set.

Subroutine EDC1 copies the contents of the Display Buffer into the area containing the String and Output Buffers. Subroutine EDD8 copies these codes back into the Display Buffer.

Subroutine FBCB, with C clear, copied the contents of the String Buffer to the Output Buffer.

Subroutine FBCB, with C set, copies the contents of the Output Buffer into the String Buffer.

## Floating-Point Arithmetic

The floating-point routines use the Basic registers located at the *following addresses:*

|  |  |
| --- | --- |
| R0: | 7A00-7A07 |
| R2: | 7A10-7A17 |
| R6: | 7A30-7A37 |

Register R0 acts as a floating-point accumulator, with R2 containing the second operand. Register R6 is used for temporary storage, except as noted. It is unaffected by the floating-point arithmetic routines.

*The 8 bytes in a floating-point register are used as follows:*

|  |  |
| --- | --- |
| Byte 0 | Exponent, in signed binary |
| Byte 1 | Sign: 00= ‘+’, 80= ‘-‘ |
| Byte 3 | 1st and 2nd mantissa digits, binary coded decimal |
| Byte 4 | 3rd and 4th mantissa digits |
| Byte 5 | 5th and 6th mantissa digits |
| Byte 6 | 7th and 8th mantissa digits |
| Byte 7 | 9th and 10th mantissa digits |
| Byte 8 | 11th and 12th mantissa digits |

## Preloading of XH and YH

In the case of certain indicated routines, XH and YH must contain 7A prior to execution of the routine.

CALL 54 (F7B0) will load XH and YH with 7A

Unless otherwise noted both XH and YH will contain 7A following execution of any of the floating-point arithmetic routines listed.

## Data transfer between R0 and fixed variables

Variable into R0: subroutine DC20. Precede by loading X with initial address of variable. Following execution XH and YH will not necessarily contain 7A.

R0 into variable: subroutine DC0C. Precede by loading X with initial address of variable. Register Y will be unchanged by this routine.

The initial address of fixed numeric variables *are as follows:*

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| A | 7900 | E | 7920 | I | 7940 | N | 7968 | R | 7988 | V | 79AB |
| B | 7908 | F | 7928 | J | 7948 | O | 7970 | S | 7990 | W | 79B0 |
| C | 7910 | G | 7930 | K | 7950 | P | 7978 | T | 7998 | X | 79B8 |
| D | 7918 | H | 7938 | L | 7958 | Q | 7980 | U | 78A0 | Y | 79C0 |
|  |  |  |  | M | 7960 |  |  |  |  | Z | 79C8 |

## Clearing of registers

Clear R0: CALL EC (F757). Register Y is unchanged by this routine.  
Clear R2: Subroutine F753. XH and YH must contain 7A.

## Data transfer between registers

R0 into R2: Call E6 (F70D).  
R0 into R6: Call 80 (F707).

R2 into R0: Call 56 (F73D). XH and YH must contain 7A.  
R2 into R6: Subroutine F701. XH and YH must contain 7A.

R6 into R0: Subroutine F737.   
R6 into R2: Call 68 (F715). XH and YH must contain 7A.

Exchange R0, R2: Call 66 (F7B9). XH and YH must contain 7A.  
Exchange R0, R6: Call 64 (F7B5). XH and YH must contain 7A.

## Data transfer between R0 and Basic Stack

The Basic stack, located 7A3C-7AFF, permits up to 24 PUSHes of R0.

PUSH R0 onto Basic stack: Subroutine DBF5  
POP R0 from Basic stack: Call 30 (DC16)

## Arithmetic Operations

All calculated results are normalized with underflow to zero; each contain a 12-digit mantissa. Calculation of trigonometric functions and their inverses is based on the current trigonometric mode setting.

If an overflow results or an illegal operation is attempted, each of these routines will terminate with flag C set, and an error code in UH. Included are ERROR 37 (overflow), ERROR 38 (division by zero), and ERROR 39 (illegal function input argument).

*Arithmetic Operations:*

|  |  |
| --- | --- |
| R0+R2 into R0: | Call F0 (EFBA) |
| R0-R2 into R0: | Subroutine EFB6 |
| R0\*R2 into R0: | Call 7E (F01A) |
| R0/R2 into R0: | Call 58 (F084) |
| R0^R2 into R0: | Subroutine F89C (Register R6 used) |
| R0\*R0 into R0: | Subroutine F019 |
| 1/R0 into R0: | Call 6E (F080). HX and YH must contain 7A.) |
| SQR(R0) into R0: | Subroutine F0E9 |
| ABS(R0) into R0: | Subroutine F597. X, Y and R2 will be unchanged |
| INT(R0) into R0: | Subroutine F5BE |
| SGN(R0) into R0: | Subroutine F59D. Y and R2 will be unchanged |
| Round R0 to 10 digits: | Subroutine F932. Y and R2 will be unchanged |
| (PI) into R0: | Subroutine F5B5. PI also stored into R2. |
| LN(R0) into R0: | Subroutine F161 |
| LOG(R0) into R0: | Subroutine F165 |
| EXP(R0) into R0: | Subroutine F1CB |
| 10^R0 into R0: | Subroutine F1D4 |
| SIN(R0) into R0: | Subroutine F3A2. Register R6 used |
| COS(R0) into R0: | Subroutine F391. Register R6 used |
| TAN(R0) into R0: | Subroutine F39E |
| ASN(R0) into R0: | Subroutine F49A. Register R6 used |
| ACS(R0) into R0: | Subroutine F492. Register R6 used |
| ATN(R0) into R0: | Subroutine F496. Precede by clearing flag C |
| DEG(R0) into R0: | Subroutine F531. Register R6 used |
| DMS(R0) into R0: | Subroutine F564. Register R6 used |

## Comparisons

Subroutine D0D2 performs a comparison of R0 and R2.

*The comparison performed will depend on the contents of A as follows:*

|  |  |  |  |
| --- | --- | --- | --- |
| A=00: | R0<>R2 | A=04: | R0=R2 |
| A=01: | R0<R2 | A=05: | R0<=R2 |
| A=02: | R0>R2 | A=06: | R0>=R2 |

If the test condition is me R0 will contain 1 with flag Z clear. Otherwise, R0 will contain zero with flag Z set. Following execution YH will not necessarily contain 7A.

## Constants (From ROM table) into R2

*In each routine XH and YH must contain 7A prior to execution:*

|  |  |
| --- | --- |
| 1 into R2: | Call 6A (F88F) |
| (PI) into R2: | Subroutine F875 |
| 180/(PI) into R2: | Subroutine F866 |
| 1/(LN 10) into R2: | Subroutine F87B |
| 0.6 into R2: | Call 62 (F88B) |
| 0.9 into R2: | Subroutine F87F |
| 90 into R2: | Subroutine F883 |
| 180 into R2: | Subroutine F887 |

## Separation of integer and fraction

Call 60 (F684) will separate the integral and fractional parts of R0, placing the integral part into R0 and the fractional part into R2. Neither result will be normalized. The given value of R0 is taken to be positive. XH and YH must contain 7A prior to execution.

## Removal of sign

Call 6C (F6FB) loads the sign byte of R0 into A and makes R0 positive. XH and YH must contain 7A prior to execution.

## Normalization of R0

Either of the subroutines Call E8 (F661) or Call 52 (F663) will normalize the contents of R0. The former applies a positive sign to the result, the latter takes the contents of A as the sign byte. In both routines underflow results in zero. Overflow sets flag C with the code for ERROR 37 in register UH.

XH and YH must contain 7A prior to execution. Register Y is not changed by either routine.

## Binary to decimal conversion

A two-byte binary number contained in Register U is converted to its decimal equivalent by execution of Call 10 (DD2D). The instruction calling this subroutine must be followed by a data byte which specifies the effect produced. Two options exist: the converted value may be stored into R0 or character codes for its display may be stored into memory beginning at the address contained in register Y.

*Binary to decimal:*

|  |  |  |
| --- | --- | --- |
| Data byte | Interpolation of U | Disposition of result |
| 00 | Unsigned binary | Stored into R0 |
| 40 | Unsigned binary | Characters stored, no sign |
| 60 | Unsigned binary | Characters stored, with sign |
| 80 | Signed binary | Stored into R0 |
| C0 | Signed binary | Characters stored, no sign |
| E0 | Signed binary | Characters stored, with sign |

Register YH will not necessarily contain 7A following execution.

## Decimal to binary conversion

The binary equivalent of the contents of R0 (ignoring fractions) is calculated and stored into Register U by Call D0 (D5F9). The result is also stored into R0 (in binary format). A will equal UL.

Two data bytes must follow the instruction calling this subroutine. The first if these specifies the range of values permitted. The second is added to the return address when the contents of R0 are not within that range, in which case UH will contain the code for ERROR 19.

Tabulated below are the permitted ranges, in decimal, for given values of the first passed byte. When negative values are permitted, results are in the signed binary.

*Decimal to binary:*

|  |  |  |  |
| --- | --- | --- | --- |
| 00: | 0 to 65535 | 0A: | 0 to 155 |
| 02: | 0 to 65279 | 0C: | 0 to 80 |
| 04: | -32768 to 32767 | 0E: | 0 to 26 |
| 06: | 0 to 32767 | 10: | 0 to 25 |
| 08: | 0 to 255 |  |  |

If any of the above passed bytes is increased by 1, zero will be eliminated as a permitted value.

# 

# Binary Routines

## Storage and recall of two-byte data:

|  |  |
| --- | --- |
| Call F6 (DDB5) | Followed by data bytes nn nn, stores UH into address nnnn and UL into nnnn+1. Following execution, A will contain UL, X will contain nnnn+1. Y and U are unchanged. |
| Call F4 (DBBC) | Followed by data bytes nn nn, loads UH with the contents of nnnn and UL with the contents of nnnn+1. ‘A’ will contain the same byte as UH. X and Y are unchanged. |
| Call CA (C001) | Followed by data byte nn, stores XH into address 78nn and XL into 78nn+1. Following execution, A will contain XL, U will contain 78nn+1. X and Y are unchanged. |
| Call CC (DDC8) | Followed by data byte nn, loads XH with the contents of 78nn and XL with the contents of 78nn+1. ‘A’ will contain the same byte as XH. Y and I are unchanged. |

## Operating in binary

Subroutine DAA8 will replace U by its complement. This is equivalent to replacing U by &10000-U. X and Y are unchanged.

Subroutine DFE2 replaces U by U-X. If the result is negative, flag C is cleared with UH containing 16. X and Y are unchanged.

Call 50 (DA71) replaces Y by U\*Y, the result is also placed into X. If the result exceeds FFFF flag X is set. Contents of U are lost.

## Binary format in R0

*The following format is used for data in R0:*

|  |  |
| --- | --- |
| 7A04 | B2, identifying R0 as containing binary |
| 7A05 | First byte of data |
| 7A06 | Second byte of data |

Data in this format may not be used as input for floating-point arithmetic routines. It may, however, constitute the contents of a variable. Basic routines will correctly interpret it.

Subroutine D9E7 stores the contents of U into R0, in binary format. Following execution, A will contain UL. Y and U are unchanged.

Subroutine D9E3 stores the byte pointed to by X into R0, in binary format. Following execution, UL will contain that same byte, UH will contain 00. Y is unchanged.

# Character String Routines

## String Pointer

A string pointer in R0 has the following format:

|  |  |
| --- | --- |
| 7A04 | D0, identifying R0 as a string pointer |
| 7A05 | High byte, beginning address of string |
| 7A06 | Low byte, Beginning address of string |
| 7A07 | Number of characters in string |

Call 24 (DEAF) stores a string pointer into R0, taking X as the beginning address, and A as the length of the string. Y is unchanged.

Call DC (DEBC) loads X with the address, and UL and A with the length of the string pointed to by R0. Register Y is unchanged.

Subroutine D048 will copy the string beginning at X, containing UL characters, into memory at address Y. After execution, Y will contain the address following that of the last character.

## Equivalents of Basic functions:

1. String input, numeric output. Prior to execution R0 must contain a pointer to the string. The numeric result will be placed un R0.
   1. ASC: Subroutine D9DD. Precede by loading YL with 60. Result will be in binary format.
   2. LEN: Subroutine D9DD. Precede by loading YL with 64. Result will be in binary format.
   3. VAL: Subroutine D9D7. Result will be in decimal.
2. Numeric input, string output. Prior to execution, R0 must contain numeric data, either binary or decimal format, and 7894 must contain 10. Following execution, R0 will contain a pointer to the resulting string, and UH will contain either 00 or an error code.
   1. CHR$: Subroutine D9B1. C1 is placed into 7A04 instead of D0
   2. STR$: Subroutine D9CF

## String comparisons

Subroutine D0F9 will compare the two strings indicated by string pointers in R0 and R2, here designated as R0$ and R2$.

*The comparison performed will depend on the contents of A, as follows:*

|  |  |  |  |
| --- | --- | --- | --- |
| A=00 | R0$<>R2$ | A=04 | R0$=R2$ |
| A=01 | R0$<R2$ | A=05 | R0$<=R2$ |
| A=02 | R0$>R2$ | A=06 | R0$>=R2$ |

If the test condition is met, R0 will contain 1 with flag Z clear, otherwise R1 will contain 0 with flag Z set. R2 will be unchanged. Note: Comparisons using A=5 or 6 will not work with ROM version A01.

# Variables used by Basic

## Locating a named variable

A variable specified by its name may be located by Call 0E (D461). Preceding execution, the variable sought must be specified according to the following rules:

1. UH must contain the ASCII code for the first character, A to Z, of the name of the variable. { @, (, ), @, $ } may not be used here.
2. For one-character names, UL must contain zero. For two-character names UL must contain the ASCII code for the second character except that when the second character is 0 to 9 then UL should contain 10 to 19.
3. If a string variable is specified UL must be increased by 20, if a dimensional variable by 80.
4. When a single subscript is used 01 must be stored into 788C and R0 must contain the value of the subscript, in decimal.
5. When a double subscript is used, 02 must be stored into 788C. The first subscript, in decimal, must be pushed into the Basic data stack. The second subscript, in decimal, must be contained in R0.

Two data bytes must follow the instruction calling this subroutine. The first of these two passed bytes should be 5A. Following execution of Call 0E, U will contain the beginning address of the variable. Also, R0 will contain a variable pointer having *the following format.*

|  |  |
| --- | --- |
| 7A04 | D0 |
| 7A05 | High byte of variable address |
| 7A06 | Low byte of variable address |
| 7A07 | Length of string variable, 88 if numeric |

If a subscript is too large, or a subscripted variable was not previously dimensioned, the return address is incremented by the second passed byte, with UH containing an error code. If an non-subscripted previously undefined variable was sought, 7A05-7A06 will contain the codes for the variable name, with 80 added to the first code.

## Loading R0 with variable contents

Call 0E (D461) may be used to load R0 with the contents of a variable specified by name. Preceding execution the variable must be specified according to the same rules used above.

The first of two passed bytes must be 52. Following execution, R0 will contain numeric data or a pointer to a string, as determined by the variable specified.

If a subscript is too large, or a subscripted variable was not previously dimensioned, the return address is incremented by the second passed byte with UH containing an error code. If a non-subscripted previously undefined variable was sought, R0 will contain zero.

## Clearing variables

Subroutine D080 clears all Basic variables. No changed in Y.

Subroutine D091 clears main memory variables only, by representing the start of variable pointer (7899-789A). No change in X, Y, or U.

# TIME and RANDOM Number

## TIME (Format used by Basic)

Subroutine DE82 copies TIME into R0, in the same format used by Basic. Registers XH and YH will contain 7A following execution.

Subroutine DE1D stores R0 into TIME, with R0 given in the same format used by Basic. One data byte is passed to this subroutine. It is passed to the return address if R0 specifies TIME incorrectly.

## TIME (Alternate format)

*Data may be transferred between R0 and the timer with the following format used in R0:*

|  |  |
| --- | --- |
| 7A00 | Not used |
| 7A01 | Not used |
| 7A02 | 1st nibble=month number (hex), 2nd nibble=week-day (0-6) |
| 7A03 | Date of month (decimal) |
| 7A04 | Hour (0 to 23, decimal) |
| 7A05 | Minute (0-59, decimal) |
| 7A06 | Second (0 to 59, decimal) |
| 7A07 | Not used |

Subroutine E59A copies R0 into TIME, X and Y are unchanged.

Subroutine E5B4 copies TIME into R09, X and Y are unchanged.

## Random Number

Subroutine F5DD placed RND (R0), interpreted as in Basic, into R0. The random number is replaced by its next value. Both R2 and R6 are used, XH and YH will contain 7A following execution.

Call 5C (F61B) replaces the random number, located 7B00-7B07 in decimal format, by its next value. Precede by loading XH and YH with both R0 and R2 will be changed.

# Reading codes from Basic

## Evaluation of expressions in Basic

Call DE (D6DF) evaluates an expression specified by Basic code, beginning at the address contain in Y. The result is placed into R0. The Basic expression is terminated by any of the condes 00 (null), 0D (ENTER), 2C (comma), 3A (colon), or 3B (semicolon).

The instruction calling this subroutine must be followed by a data byte, which will be added to the return address, with UH containing an appropriate error code, if an error occurs.

Numeric results obtained form AND, OR, NOT, ASC, LEN, POINT, PEEK or PEEK# are in signed binary, other numeric results are in decimal. String results are expressed as string pointers in R0, with the string itself located in the String Buffer.

## Conversion of hex characters to byte

Subroutine ED95 loads A with a byte determined by a pair of codes for hex digits, located starting at address X. Codes for characters other then 0-9 or A-E will clear flag C, signaling an error. After execution X will point to the next address, Y and U are unchanged.

# Search for Basic program line

Subroutine D2EA searches for the address in memory of the Basic line whose number, in hexadecimal, contained in U. The search will begin at the address indicated by the START OF BAIC, or START OF BASIC in ROM, pointer. It will end if a stop byte is encountered.

The same subroutine will search for the line with a specified label. In this case, UH must contain FF, and R0 must contain a string pointer, pointing to the string forming the label sought. If necessary, a search for a label will bypass stop bytes, continuing until END OF BASIC is reached.

In either case, if the line sought is located, the address of the first statement in that line is placed into 789A6-A7. The address at which the line’s number begins is less than that address by 3.

A data byte must follow the instruction calling this subroutine. If the line sought is not found this data byte is added to the return address, with UH containing 0B.

As an alternative, a search for a specified line number, but not for a label, may be begun at the address pointed to be START OF EDIT. This is done by subroutine D2E0, which is identical in all other respects to the above subroutine.

# Keyboard

## Poll of keyboard

Subroutine E42C loads A with the code for the key, other than BREAK, that is currently depressed. If no key is depressed A will be loaded with zero. Register Y is unchanged by the subroutine.

*The codes determined by the keys are as follows:*

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 01 | SHIFT | 0F | OFF | 19 | RCL | 2D | - | 35 | 5 | 43 | C | 4B | K | 53 | S |
| 02 | SML | 11 | F1 | 1B | DEF | 2E | . | 36 | 6 | 44 | D | 4C | L | 54 | T |
| 08 | Left | 12 | F2 | 1F | MODE | 2F | / | 37 | 7 | 45 | E | 4D | M | 55 | U |
| 09 | Template | 13 | F3 | 20 | SPACE | 30 | 0 | 38 | 8 | 46 | F | 4E | N | 56 | V |
| 0A | Down | 14 | F4 | 28 | ( | 31 | 1 | 39 | 9 | 47 | G | 4F | O | 57 | W |
| 0B | Up | 15 | F5 | 29 | ) | 32 | 2 | 3D | = | 48 | H | 50 | P | 58 | X |
| 0C | Right | 16 | F6 | 2A | \* | 33 | 3 | 41 | A | 49 | I | 51 | Q | 59 | Y |
| 0D | ENTER | 18 | CL | 2B | + | 34 | 4 | 42 | B | 4A | J | 52 | R | 5A | Z |

## Wait for input form keyboard

Subroutine E243 polls the keyboard until a key code is obtained. Except for the SHIFT, SML, and DEF keys, which act as prefixes, the use of a key produces a return, with a character code in A. The 7-minute power down timer operates during the wait. Register Y is unchanged.

The codes obtained for display characters are their ASCII codes.

The codes for control and editing keys requiring the SHIFT prefix are *as follows:*

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 1A | CA | 1C | INS | 1D | DEL | 1E | SHIFT / MODE |

The codes for the alphabet keys prefixed by DEF are their ASCII codes plus 40. The other keys accepting the DEF prefix use *these codes:*

|  |  |  |  |
| --- | --- | --- | --- |
| 80 | DEF SPACE | 9D | DEF = |

The code for BREAK is 0E. If BREAK has been keyed, subroutine E243 will continue to return with the code A each time it is called, until bit 1 of address F00B in the I/O port is cleared.

## Detection of BREAK key

Call A6 (E451) will clear flag Z if BREAK has been keyed. There is no effect on the CPU registers or on other flags.

If BREAK has been keyed, this subroutine will continue to return with flag Z cleared each time it is called, until bit 1 of address F00B in the I/O Port is cleared.

# Display

## Cursor Pointer

The contents of the Cursor Pointer (7875) specify a column of dots in the LCD display. Columns are numbered from 00-9B (0-155 decimal).

## To clear display

Call E2 (EE71) clears the display. Registers X and Y are unchanged.

## To display data in R0 (or string pointed to by R0)

Subroutine E8CA may be used to display numeric data contained in R0, given in decimal or binary, or a string pointed to by R0. Display will be in default format. Strings are displayed left-justified and numeric data right-justified. Precede by storing 20 as the value of the display parameter (7880).

## To display numeric contents of R0, formatted

The display will be formed beginning at the location pointed to by the Cursor Pointer (7875). Contents of R0 must be in decimal.

1. Store format specifications as follows
   1. 7895: Editing characters. 10=comma separation, 20=forced sign, 40=asterisk fill, 80=scientific format
   2. 7896: Number of characters preceding the decimal point, including spaces for sign, and commas when specified.
   3. 7898: Number of characters including and following decimal point
2. Load A with 00, 01, or 02, to specify the position of the characters to display according to the following
   1. A=00: Data displayed right-justified in 13-character block
   2. A:01: Data displayed left-justified in block of whatever size is required, up to a maximum of 16 characters.
   3. A=02: Data displayed right justified in 26-character block.
   4. Then execute Call 96 (EA78) to form a string of characters, in the area 7A10-7A34, to represent the data to be displayed
3. Execute the instruction STX U, then execute Call 92 (ED00

The cursor pointer will be updated to point to the next available display position. Flag C will be set it the display has been filled.

## To display a string pointed to be R0

Execute Call DC (DEBC) followed by Call 92 (ED00). The cursor pointer will be updated to the point to the next available display position. Flag C will be set if the display has been filled.

## To display contents of output buffer

Subroutine ECFA displays the contents of the Output Buffer staring at 7B60, ending when the display is filled. Characters are entered into the display beginning at the position specified by the Cursor Pointer.

## To display a sequence of characters

Subroutine ED3B will be display a sequence of characters whose codes are located anywhere in memory. Precede by loading U with the beginning address of the stored codes, and XL with the number of codes. The characters will be entered into the display beginning at its left end.

## To display a program line

The line to be displayed must be put into the Input Buffer, starting with its two-byte line number, omitting the link byte, and continuing with the code for the Basic statements in the line. A line may be placed into the Input Buffer in that form in any of the following ways:

1. Subroutine D262f may be used to place the first Basic line into the Input Buffer, the SEARCH pointer will be set to point to that line.
2. Subroutine D2EA may be used to locate the line whose number has been loaded into U. A data byte, which will be added to the return address if the specified line does not exist, must follow the instruction calling this subroutine. After execution of subroutine D2EA, load XH and XL with the contents of 78A6 and 78A7, decrement X twice and execute subroutine D2D0. The specified line will be in the Input Buffer, and the SEARCH pointer will point to that line.
3. The program line that follows the one currently pointed to by SEARCH may be placed into the Input Buffer, with automatic updating o the SEARCH pointer by execution of subroutine D2B3.

After the Basic line has been placed into the Input Buffer, it is displayed by execution of subroutine E8CA, provided that an appropriate value had been stored as the Display Parameter (7880). With 10 as that value, the line is displayed from its start, without a colon following the line number in the display, with 14, the colon will be included.

The cursor will be included in the display if the Display Parameter contains 50 (no colon) or 54 (colon included). Register Y must be loaded with the address, in Input Buffer, at which the cursor is to appear.

## To display the contents of input buffer with tokens expanded

Subroutine E8CA, with Display Parameter (7880) containing zero, displays the contents of the Input Buffer, starting at 7BB0, ending when 0D code is reached. The Input Buffer may be filled with 0D codes preceding this, by execution of subroutine D02B.

The cursor will be included in the display if 40 is used as the Display Parameter value. Register Y must be loaded with the address, in Input Buffer, at which the cursor is to appear.

## To display a single character

Subroutine ED4D enters the character whose code is in A into the display, at the position pointed to by the Cursor Pointer (7875). The Cursor Pointer will be incremented by 6 each tie a character is added to the display, pointing to the next character position. If the right end of the display is reached, additional characters wrap around to the eft end of the display.

The character whose code is in A may also be placed into the display by subroutine ED57. In this case there will be no automatic increment of the Cursor Pointer.

## Graphic Display

Subroutine EDEF displays the dots specified by the contents of A in the column pointed to by the Cursor Pointer. The Cursor Pointer is not incremented automatically. Call 8E (EDB1) may be used to do so.

The dots displayed are determined by the *bits contained in A:*

|  |  |  |  |
| --- | --- | --- | --- |
| 01 | Top Dot | 10 | 5th |
| 02 | 2nd | 20 | 6th |
| 04 | 3rd | 40 | Bottom |
| 08 | 4th |  |  |

## Timed Delay

Call AC (E88C) produces a delay, equal in duration to the product of a 1/64th second cy the contents of Register U. If the BREAK key is pressed during execution of this routine the timing cycle ends and there is a return with flag C set. Register Y is not affected.

This subroutine may be used to produce the equivalent of a PAUSE by effecting a temporary delay while information is being displayed.

## BEEP

Subroutine E669 is equivalent to BEEP 1. It ignores BEEP ON/OFF.

Subroutine E66F produces a tone with the pith determined by UL, and duration determined by X. BEEP ON/OFF is ignored. The frequency of the tone is approximately 59230/(7.47+UL) CPS (hz?) (calculated in decimal). The duration, number of cycles, is &100 less than the contents of X.

## Power Down

Subroutine CD71 is equivalent to OFF.

Subroutine E33F is equivalent to the automatic 7-minute power down.

## Termination at ‘READY’

A machine language program may be terminated by setting the computer to ‘Ready’ mode, in which it is awaiting further instructions form the keyboard.

## Terminate with display

Call 46 (CA8D) sets ‘Ready’ mode, retaining the display contents.

If subroutine E8CA has been used to produce a display with the cursor contained in it, the cursor left and right keys will be enabled. In other cases, however, use of the cursor control keys may result in a meaningless display.

## Terminate with prompt mark

Call 42 (CA58) will place the computer in ‘Ready’ mode, with the prompt mark displayed.

## Terminate with error message

The following routines will terminate a machine language program with an ERROR message displayed in ‘Ready’ mode:

1. Call E0 (CD8B) displays ERROR N, where n is contained in UH
2. Call E4 (CD89) displays ERROR 1

When the ERROR message is displayed, by the up arrow key may be used to display a Basic line at the address pointed to but Y. A meaningless display will result unless an effort has been made to load Y to point to the desired location.

Register Y can be made to point to the location in Basic from which the machine language program was called by either of the following:

1. Provided that the stack has been kept balanced. POP Y twice.
2. Load register S with 784A, then execute POP Y.

# Printer Routines

## Motor Off

Subroutine A769 is required following certain printer subroutines.

*Equivalents of certain Basic statements:*

|  |  |
| --- | --- |
| TEXT | Subroutines ACBB, A9BB, A9D5, A769 |
| GRAPH | Subroutines A9D5, A769, ABEF, store FF into 79F0 |
| CSIZE n | Store into 79F4 |
| COLOR n | Subroutine A519; Precede by loading UL with n (0 to 3). Follow by subroutine A769 (Motor off) |
| LF n | Subroutine AA04. Precede by storing n (in signed binary) into 7B7F-7B80, and loading XH with 7B, XL with 7F.Follow by subroutine A769 (Motor off) |
| LCURSOR n | Subroutine AC52. Precede by loading XH with FF, and XL with n. Follow by subroutine A769 (Motor Off) |
| SQRGN | Subroutine AC97 |
| ROTATE n | Store n (0 to 3) into 79F2 |

## Printing in text mode

1. To print contents of R0 (or string pointed to by R0), using default format:
   1. Store 04 into 788E, execute subroutine B3EA
   2. Numeric data is printed right-justified, strings left-justified. Printer advances to next line if necessary. If CSIZE set is larger than 2, it will be changed to 2. TEXT mode will be set. A carriage return, with line feed, will follow printing.
2. To print contents of R0, or string pointed to, formatted:
   1. Store format specifications into memory, as follows:
      1. 7BAA: Editing characters. 10=comma separation, 20=forced sign, 40=asterisk fill, 80=scientific format
      2. 7BAB: Number of characters preceding decimal point, including spaces for sign and commas (when specified)
      3. 7BAC: Maximum number of characters to be printed, in printing of character strings; if no limitation, zero
      4. 7BAD: Number of characters including and following decimal point
   2. Store zero into 78EA and 79F5
   3. Execute subroutine B49A (Print); optionally subroutine A9F1 (CR with line feed); subroutine A769 (Motor Off)
3. To print contents of R0 (or string pointed to by R0), formatted, starting at current pen location, with line advance when necessary:
   1. Store format specifications into the 7BAA-AD, as described above
   2. Store 00 into 79EA, 08 into 79F5
   3. Execute subroutine B49A. Follow by subroutine A769 (Motor off)
4. To print a single character (no carriage return or line feed)
   1. Store the ASCII code of the character to be printed into 7B7F
   2. Load XH with 7B, XL with 7F. Store zero into 79EA.
   3. Execute subroutine A78a. Follow by subroutine A769 (Motor off)
5. To execute carriage return with line feed:
   1. Load XH with 7B, XL with 7F. Execute subroutines A9F1, A769

## Plotting (GRAPH mode)

1. To move pen, using given x and y increments:
   1. Specifying increments in two-byte signed binary, store the x-increment into the 7B7E-7B7F, and the y-increment into 7B7C-7B7D
   2. Load XH with 7B, XL with 7C
   3. Execute subroutine AD78. Follow by subroutine A769 (Motor off)
2. To move pen to location having coordinates (x,y):
   1. Specifying x and y in two-byte signed binary, store x into 7B7E-7V7F, and y into 7B7C-7B7C-7B7D
   2. Load XH with 7B, XL with 7C
   3. Execute subroutine AC07. Follow by subroutine A769 (Motor off)
3. To draw a line, starting from current pen location, using given increments in x and y:
   1. Specifying increments in two=byte signed binary, store the x-increments into 7B7E-7B7F, and the y-increment into 7B7C-7B7D
   2. Load XH with 7B, XL with 7C. Store FF into 79E9: store line type (0 to 0) into 79EA. Store 01 into 7B84
   3. Execute subroutine AD07 (Dar Line) and A769 (Motor off)
4. To draw a square, taking current pen location as lower left corner:
   1. Load A with length of a side of the square
   2. Load XH with 7B, XL with 7F. Store line type (0-9) into 79EA
   3. Execute subroutine AC2F (Draw square) and A769 (Motor off)
5. To draw a rectangle, beginning at current pen location
   1. Specifying dimensions in two-byte signed binary, store the x-dimension into 7B7E-7B7F and the y-dimension into 7B7C-7B7D
   2. Load XH with 7B, XL with 7C. Store FF into 79E9. Store line type (0 to 9) into the 79EA

## Printing in GRAPH mode

1. To print contents of R0 (or string pointed to by R0), formatted, starting from present pen location, in direction specified by ROTATE:
   1. Store format specification int 7BAA-AD, as in TEXT mode
   2. Store zero into the e79EA and 79F5
   3. Execute subroutine B433. Follow by subroutine A769 (Motor off)
2. To print a single character, in direction specified by ROTATE:
   1. Store the ASCII code of the character to be printed into 7BFF
   2. Load XH with 7B, XL with 7F. Store zero into 79EA
   3. Execute subroutine A781. Follow by subroutine A769 (Motor off)

## Printing program lines (Text mode)

1. To place a program line into the Input Buffer prior to printing the line to be printed must be put into the Input Buffer, starting with its two-byte line number, omitting the link byte, and continuing with the codes for the Basic statements in the line. A line put into the Input Buffer in that form in any of the following ways:
   1. Subroutine D26F ay be used to place the first Basic line into the Input Buffer: the SEARCH pointer is set to point to that line.
   2. Subroutine D2EA may be used to locate the line whose number has been loaded into Register U. A data byte, which will be added to the return address if the specified linen does not exist, must follow the instruction calling this subroutine. After execution of subroutine D2EA, load XH and XL with the contents of 78A6 and 78A7, decrement X twice, and execute subroutine D2D0. The specified line will be in the Input Buffer, and the SEARCH pointer will point to that line.
   3. The program line that follows the one currently pointed to by SEARCH may be placed into the Input Buffer, with automatic update of the SEARCH pointer, be execution of subroutine D2B3.
2. To print the line stored in the Input Buffer:
   1. Execute subroutine AE3A will perform a line feed with carriage return, then print the line contained in the Input Buffer. Follow by subroutine A769 (Motor off)

## Miscellaneous (Mechanical)

1. Paper Feed Key Disable / Enable
   1. Disable: Subroutine A306
   2. Enable: Subroutine A30A
2. To move paper (no update of coordinates, GRAPH mode):
   1. With n a two-byte signed binary number, subroutine AA0E feeds the paper by n graphic units. Negative will feed in reverse. Precede by loading XH with 7B, XL with 7F, Y with n, and UL with 01.
3. Pen Up/Down control:
   1. Pen Up: Subroutine AB08
   2. Pen Down: Subroutine AAFA
   3. Pen Up/Down: Subroutine AAE3 sets pen up if 79E9 contains zero, down if 79E9 contains FF
4. To move pen (no update of coordinates, GRAPH mode):
   1. Move right, n graphic units, with subroutine A28E. Precede by loading XL with n. Follow by subroutine A769 (Motor off)
   2. Move left, n graphic units, with subroutine A28B. Precede by loading XL with n. Follow by subroutine A769 (Motor off)
   3. Return pen to left end, without line feed, with subroutine A9D5. Follow by subroutine A769 (Motor off).
5. To cycle pen to next color:
   1. Execute subroutine A629. Follow by subroutine A769 (Motor off)

# Cassette Routines

## To save a file on cassette

1. Construct header in Output Buffer area (7B60-7BAF)
   1. Execute subroutine BBD6 to form Lead-in, including file type code. Precede by loading A with file type code: 00=ML, 01=Basic, 02=Reserve memory, 04=data file. Others definable by user.
   2. File name is optional. If to be included its character codes must be stored into addresses 7B69-7B78 (maximum 16 characters)
   3. Store beginning address of file being saved, into 7B82-7B83. Store 1 less than the number of bytes in the file, into 7B84-7B85
   4. If a ML program (file type 00) is being recorded, store the beginning execution address, or default value of FFFF, into 7B87-87
2. Record Header onto cassette
   1. Store value of Cassette Parameter into 7879: 00=RMT 0, 10=RMT 1
   2. Execute Call B0 (BCE8)
3. Record file
   1. Load X with beginning address (obtainable from 7B82-83). Load I with 1 less than number of bytes (obtainable from 7B84-85)
   2. Execute Call AA (BD3C). Terminate operation with Call B4 (BBF5)

## To load a file form cassette

1. Construct header in Output Buffer area, 7B60-7BAF
   1. Load A with code for file type sought
   2. Execute subroutine BBD6
   3. Store (optional) file name into 7B69-7A
2. Find file on cassette
   1. Store value of Cassette Parameter into 7879: 00=RMT, 90=RMT 1
   2. Execute Call B0 (BCE8). A search will begin for a previously recorded header having the file type specified. If found, its file name is displayed and stored into 7B91-A0. If this name differs from the one specified, the search continues.
   3. If flag C is set, there was a checksum error, end with Call B4
3. Load file into computer
   1. Load X with the address into which the file is to be loaded and load U with 1 less than the number of bytes to be loaded. The values for these parameters that ere originally recorded are not obtainable, respectively, from 7BAA-AB and 7BAC-AD
   2. Execute Call AA (BD3C). Following loading, register X will contain the address immediately the last address loaded.
   3. If flag C is set, there was a checksum error, end with Call B4.
4. Terminate operation of the recorder with Call B4 (BBF5)

## Control of relays

Subroutine BF28 may be used to operate the remote control relays. The effect of this subroutine is determined by the *contents of A:*

|  |  |
| --- | --- |
| A=03 | Close relay, REM 0 |
| A=05 | Open relay, REM 0 |
|  |  |
| A=09 | Close relay, REM 1 |
| A=11 | Open relay, REM 1 |

## Record, Load, or verify block of data

Register X must be loaded with the beginning address of the block of memory involved and register U must contain 1 less than the number of bytes in that block. Then Call AA (BD3C) will perform the operation specified by the contents of the Cassette Parameter (7879), as follows:

* When 7879 contains 00, the block of data is recorded onto tape.
* When 7879 contains 80, the block of data read from tape is transferred to memory. If a checksum error occurs, there will be a return with flag C set.
* When 7879 contains 40, data read from tape is compared to the data in the block of memory, without any transfer of data. If a discrepancy is found there is a return with flags C and V set. If a checksum error occurs a return with flags C and H set.

In each of the above cases, Call B4 (BBF5) may be used to terminate the operation. It will open the relay, shutting off the recorder.

## Record or load a single byte

Subroutine BDCC will record the byte contained in the accumulator.

Call A4 (BDF0) will read one byte from tape and load it into the accumulator. This routine also checks whether the BREAK key has been used, if is has flag C is set.