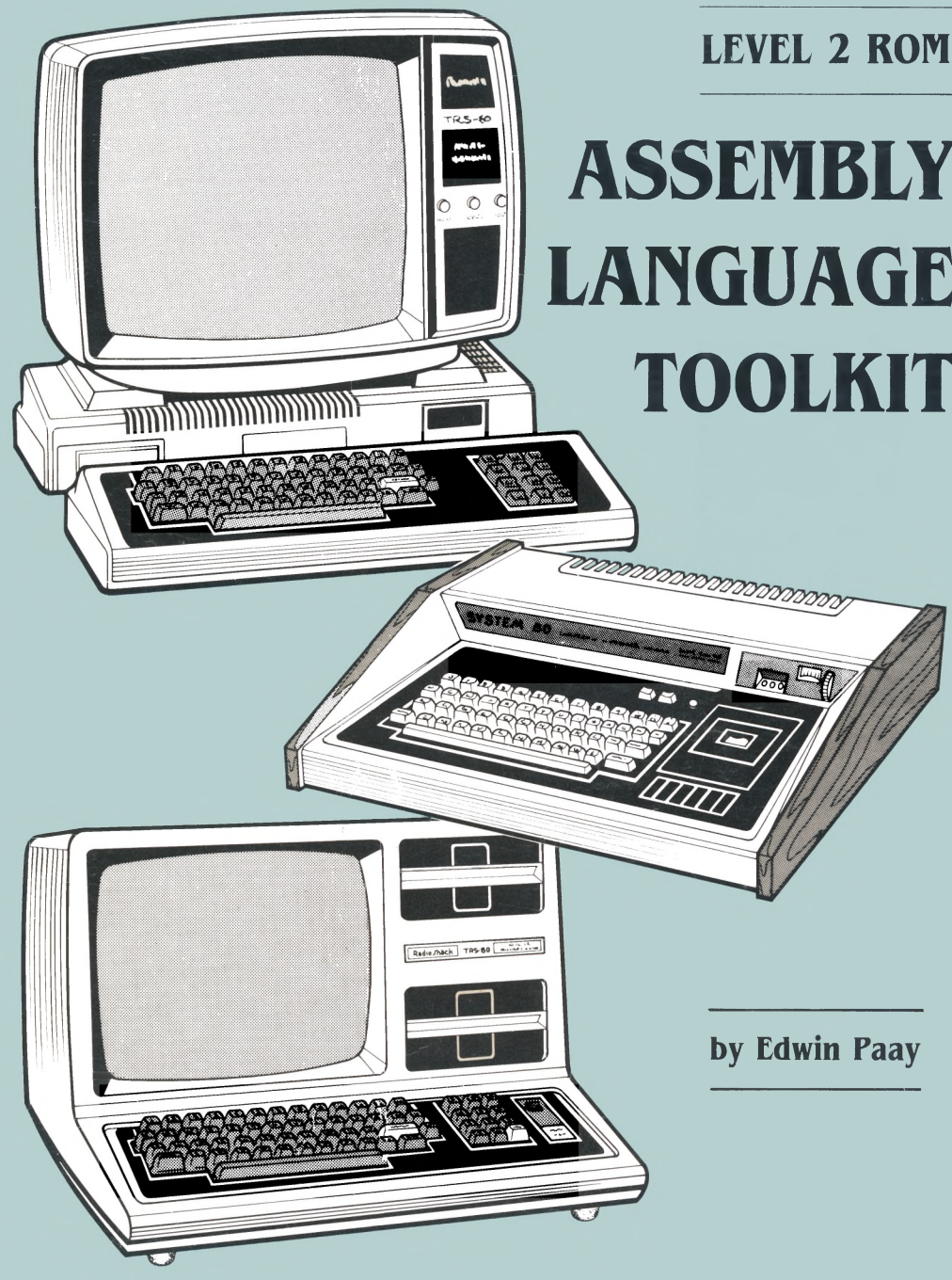
LEVEL 2 ROM

# ASSEMBLY LANGUAGE TOOLKIT




by Edwin Paay

FOR TRS-80™ MODEL 1, MODEL 3
PMC-80™/VIDEO GENIE™/SYSTEM 80™

**About the Author:**

Edwin Paay is one of Australia's leading exponents in assembly language programming on the TRS-80 Model I Microcomputer. Largely self-taught, he has developed many useful utility programs and written many articles to assist TRS-80 owners obtain the maximum use from their computers. The Level 2 ROM Assembly Language Toolkit follows on from his earlier book "Level 2 ROM Reference Manual" (now out of print) which, at the time, was the leading reference manual for assembly language programmers wishing to make use of ROM routines in their own Programs.

Edwin has been technical manager for Micro-80 for the past two years where he has developed numerous hardware interfaces and improvements for the TRS-80 and other computers.

**About the Publishers:**

Micro-80 Pty. Ltd., publishes the leading specialist magazine for TRS-80/Video Genie/System 80 users, in both Australia and the United Kingdom. This monthly magazine contains new programs, hardware articles, instructional articles for programmes and many useful hints and tips. This Level 2 ROM assembly language toolkit has been published in response to requests from many readers for assistance in using ROM routines in their own programs.

**Level 2 ROM Assembly Language Toolkit — first edition 1982**

**Published by:**
MICRO-80 Pty. Ltd.
433 Morphett Street, Adelaide,
South Australia. 5000
Tel. (08) 211 7244

**Printed by:**
G. F. STEVENSON PRINTERS,
36 Sturt Street, Adelaide,
South Australia. 5000

## WHO SHOULD BUY THIS PACKAGE?

The LEVEL 2 ROM ASSEMBLY LANGUAGE TOOLKIT is intended for users of TRS-80 Model 1 and 3 and SYSTEM 80/VIDEO GENIE/PMC-80 microcomputers who are interested in writing machine language programs, at whatever level of experience. It will enable the novice machine language programmer to quickly and simply make his machine perform useful functions in machine language, and to successfully debug his or her programs using the flexible DBUG program included in this package. On the other hand, the experienced machine language programmer who uses this package will find he is writing shorter, more elegant programs to achieve the desired result.

The BASIC programmer too, will find this manual helpful. It will give him a much better understanding of the way in which his BASIC programs use the computers memory space and will help him write machine language programs which interface with BASIC in an efficient and professional manner. In short, if you use a TRS-80 (tm), then you need this assembly language development system.

COPYRIGHT (C) 1980 - 1981 Edwin. R. PAAY - ALL RIGHTS RESERVED. No part of this manual may be reproduced, stored in a retrieval system or transmitted in any form or by any means including photocopying, electronic, mechanical or otherwise without written permission from the publisher.

The proprietary rights to the BASIC interpreter which is the subject of the ROM reference section of this package, are held by MICROSOFT. The author and publisher of this manual are in no way attempting to infringe these rights. The purpose of this assembly language aid, is to assist users of TRS-80 (tm) microcomputers in using and understanding their machines to the full.

*** SPECIAL NOTICE ***

The author and MICRO-80 PRODUCTS assume no liability with respect to the use of this publication nor any damages arising from the use of any information contained herein. This package is sold in an "as is" condition and is not represented as being free from errors. (TRS-80 is a trade mark of the TANDY CORPORATION.)

***** TABLE OF CONTENTS *****

*** PART ONE ***

-0000000000-

## - LEVEL II ROM REFERENCE SECTION -

### --** INTRODUCTION **--

The BASIC interpreter in the TRS-80 (tm) microcomputer is a machine language program which resides in a read only memory (ROM) from address 0000 Hex to address 2FFF Hex.in the Model 1 or to 377F Hex. in the Model 3. The routines in this interpreter have never been officially revealed. Other publishers have released books on the Model 1 interpreter which are interesting to read, but not very practical for the assembly language programmer who wants to know where useful routines are stored, which can be used as subroutine CALLs. To our knowledge no one has released an assembly language development package such as this until now.

This manual reveals most usable routines. The routines are put together in tables, which makes it easy to locate a routine when needed. The secrets of the reserved memory are laid bare and best of all, there are sample programs to illustrate how you can make use of these routines to simplify and enhance your own machine language programs.

The LEVEL 2 ROM ASSEMBLY LANGUAGE TOOLKIT is in three parts. The first two parts are the ROM REFERENCE SECTION while the third part contains the operating instructions for the DBUG program supplied on tape as part of this package.

Part 1 explains in detail how to use the ROM routines. It contains tables indexed by function so that you can quickly locate the routine which is best suited to your particular purpose. It also contains sample programs which illustrate how the routines may be used in your own machine language programs.

Part 1 also contains discussion on the ACC, AACC and the NTF which are quite often mentioned in the text. It is recommended that these sections be read first so that the reader understands the meaning of these terms. (Note that the A register contained in the Z80 is always referred to as the A register and NEVER as ACCUMULATOR in the text. This is done to avoid confusing it with the area in memory referred to as ACCumulator).

Part 2 contains an extended memory map which lists all usable routines and shows where the various BASIC commands are located in ROM. This serves as a useful reference to specific addresses in ROM and also as an adjunct to a disassembled listing.

Part 3 contains detailed operating instructions for DBUG, a symbolic debugger which operates from disk or tape. DBUG enables the user to quickly and easily locate bugs in his programs and fix them with the aid of single-stepping, dynamic disassembly, memory editing, examination and modification of Register, contents, setting break points etc. etc..

In preparing this manual, we have assumed that our readers understand the Z80 instruction set and have some experience of programming with the aid of an assembler. This manual is in no way a substitute for a good text on Z80 assembly language programming, rather, it will help to make assembly language programming easier and less tedious.

## - LEVEL II ROM REFERENCE SECTION -

### ***** PART 1 *****

In this part of the manual, the ROM routines which are described in numeric order in Part 2 are organised by function in a series of tables. This should enable the user to quickly choose the most suitable routine for his purpose. Each routine listed in this section also has a description in Part 2, it is recommended to read both, as often further information can be found about a routine that way. Some addresses vary between the Models 1 and 3, or do not exist in the Model 3. Where this happens, either the Model 3 address will be mentioned in the text or it will be made clear that the address applies to Model 1 or 3 only.

In interpreting these tables, the following points should be considered:

- if a register is not mentioned specifically, then the routine may alter or destroy its contents.

- the addresses given in the Tables are the entry points to the routines.

* * * * * * * * * * * * * * * * * *

## -- ARITHMETIC --

The arithmetic routines come in three main types: Integer, single or double-precision. Some information is required to use these routines. First there is the NTF, this is the Number Type Flag. It is located at 40AFH in memory and is used by BASIC when it wants to know what type of number is residing in the ACC area. The NTF will contain 2 if Integer, 4 if single-precision, 8 if double-precision or 3 if string.

Then there is the ACCumulator area (ACC). This is the area in memory from 411DH to 4124H, (See Table 1). It is used to store values and the results for most of the arithemtic routines. The type of number contained in the ACC can be determined at any time by testing the NTF. If you are loading a number into the ACC manually the NTF must also be set in order to reflect the number type. Note that the NTF actually contains the data length in all cases except with strings. (Discussed later).

Finally, there is the Auxiliary ACCumulator (AACC). This is the area 4127H to 412EH in memory. It is only used by the double-precision arithmetic routines. Note that the ACC and AACC are changed by data conversion routines but not by compares.

TABLE 1

ORGANIZATION OF ACC AND AACC

| | ADDRESS | INTEGER | SINGLE PRECISION | DOUBLE PRECISION |
|---|---|---|---|---|
| ACC | 411DH | | | LSB |
| | 411EH | | | LSB |
| | 411FH | | | LSB |
| | 4120H | | | LSB |
| | 4121H | LSB | LSB | LSB |
| | 4122H | MSB | LSB | LSB |
| | 4123H | | MSB | MSB |
| | 4124H | | EXP | EXP |
| AACC | 4127H | LSB | LSB | LSB |
| | 4128H | MSB | LSB | LSB |
| | 4129H | MSB | MSB | LSB |
| | 412AH | | EXP | LSB |
| | 412BH | | | LSB |
| | 412CH | | | LSB |
| | 412DH | | | MSB |
| | 412EH | | | EXP |

In all the values listed in Table 1, the numbers are stored as signed numbers. This means that the most significant bit in the MSB of the value is used as a sign bit (1 if number is negative or 0 if positive). The EXPonents are stored in normalized form. The true EXP is 128 less than the actual number. For more information see the Level II manual pages 8/8 to 8/10. The BC and DE registers are used extensively by the single-precision routines for one of the operands. The MSB must be in the B reg. and the LSB must be in the E reg. (4 bytes total).

When it is necessary to CALL an arithmetic function and one or both of the numbers are of the wrong type the CINT, CSNG or CDBL routines can be CALLed. These routines are found in Table 3. All arithmetic functions return with the result in the ACC. The integer arithmetic functions however, return with the result in both the ACC and the HL reg. pair. If overflow occurs on any of the integer arithmetic functions the result will automatically be a single-precision number. The NTF can be tested to see if such is the case. If any errors occur during the arithmetic operations, such as divide by zero, an error message will be printed and control will be passed to BASIC. Finally, make sure there is enough stack space for the routines to use.

TABLE 2

ARITHMETIC ROUTINES

| ADDRESS | OPERATION | INPUT | OUTPUT | OUTP. TYPE |
|---|---|---|---|---|
| | --** SINGLE-PRECISION **-- | | | |
| 070BH | ACC=(HL)+ACC | ACC,(HL) | ACC | NTF=4,SNG |
| 0710H | ACC=(HL)-ACC | ACC,(HL) | ACC | NTF=4,SNG |
| 0713H | ACC=BCDE-ACC | ACC,BCDE | ACC | NTF=4,SNG |
| 0716H | ACC=BCDE+ACC | ACC,BCDE | ACC | NTF=4,SNG |
| 0847H | ACC=BCDE*ACC | ACC,BCDE | ACC | NTF=4,SNG |
| 08A2H | ACC=BCDE/ACC | ACC,BCDE | ACC | NTF=4,SNG |
| | --** INTEGER. **-- | | | |
| 0BC7H | ACC=DE-HL | DE, HL | ACC,HL | NTF=2,INT |
| 0BD2H | ACC=DE+HL | DE, HL | ACC,HL | NTF=2,INT |
| 0BF2H | ACC=DE*HL | DE, HL | ACC,HL | NTF=2,INT |
| 2490H | ACC=DE/HL | DE, HL | ACC,HL | NTF=4,SNG |
| | --** DOUBLE PRECISION. **-- | | | |
| 0C70H | ACC=ACC-AACC | ACC,AACC | ACC | NTF=8,DBL |
| 0C77H | ACC=ACC+AACC | ACC,AACC | ACC | NTF=8,DBL |
| 0DA1H | ACC=ACC*AACC | ACC,AACC | ACC | NTF=8,DBL |
| 0DE5H | ACC=ACC/AACC | ACC,AACC | ACC | NTF=8,DBL |

Note: All registers are used.
SNG = Single precision.
INT = Integer.
DBL = Double precision.

TABLE 3.

ARITHMETIC FUNCTIONS AND NUMBER CONVERSIONS

| ADDRESS | | FUNCTION | INPUT | OUTPUT |
|---|---|---|---|---|
| 0809H | | LOG (ACC) | ACC,NTF=4 | ACC, NTF=4 |
| 0977H | & | ABS (ACC) | ACC,NTF=2,4 or 8 | ACC, NTF=2,4 or 8 |
| 0982H | $# | NEG (ACC) | ACC,NTF=4 or 8 | ACC (Negate) |
| 098AH | | SGN (ACC) | ACC,NTF=2,4 or 8 | ACC=-1,0,1, NTF=2 |
| 0A7FH | | CINT(ACC) | ACC,NTF=2,4 or 8 | ACC and HL, NTF=2 |
| 0AB1H | | CSNG(ACC) | ACC,NTF=2,4 or 8 | ACC, NTF=4 |
| 0ADBH | | CDBL(ACC) | ACC,NTF=2,4 or 8 | ACC, NTF=8 |
| 0B26H | * | FIX (ACC) | ACC,NTF=2,4 or 8 | ACC, NTF=2 or 4 |
| 0B37H | * | INT (ACC) | ACC,NTF=2,4 or 8 | ACC, NTF=2 or 4 |
| 0C51H | # | NEG (HL) | HL | ACC and HL, NTF=2 |
| 13E7H | | SQR (ACC) | ACC,NTF=4 | ACC, NTF=4 |
| 1439H | | EXP (ACC) | ACC,NTF=4 | ACC, NTF=4 |
| 14C9H | | RND (ACC) | ACC,NTF=2,4 or 8 | ACC, NTF=4 |
| 1541H | | COS (ACC) | ACC,NTF=4 | ACC, NTF=4 |
| 1547H | | SIN (ACC) | ACC,NTF=4 | ACC, NTF=4 |
| 15A8H | | TAN (ACC) | ACC,NTF=4 | ACC, NTF=4 |
| 15BDH | | ATN (ACC) | ACC,NTF=4 | ACC, NTF=4 |
| 2B05H | % | INT (ACC) | ACC,NTF=2,4 or 8 | DE |

* If the operand is smaller than 32767 then NTF=2

$ BC and DE unchanged.

\# These routines change the sign of the operand.

& Remember that NTF=2 is integer, NTF=2,4 or 8 means that the routine is suitable for integer, single and double-precision. For the above routines, the user must supply arguments which stay within the range allowed by BASIC for these functions. Check too, that the arguments have the correct number type

TABLE 3.

ARITHMETIC FUNCTIONS AND NUMBER CONVERSIONS

(Continued)

before a particular function is called as some of the functions do not check the NTF and errors could result.

% The Z flag is set if the result in DE is equal to or smaller than 255 (FFH).

## --** DATA MOVEMENT **--

This section handles the movement of data to and from memory and ACC etc. Data moved into the ACC, AACC and BCDE registers will be of the correct format only if the format of the source was correct. Some of these routines are suitable to move data from a variable created by BASIC into the BCDE registers or ACC, so that the data can be operated on by a machine language program. All usable routines are shown in Table 4.

The column labelled "ADDRESS" is the call address, the column labelled "FROM" is the source (usually a register pair is used to point to a location in memory) and the column labelled "TO" is the destination. The column labelled "BYTES MOVED" shows the number of bytes moved or the register whose value determines the number of bytes which will be moved. If applicable, the limits are also shown.

Finally, the column labelled "POINTERS" shows the way in which the pointers are changed after execution, and/or which registers are unchanged (saved) by the operation. (N/A = not applicable).

TABLE 4

--** DATA MOVEMENT **--

| ADDRESS | FROM | TO | BYTES MOVED | POINTERS |
|---|---|---|---|---|
| 09A4H * | ACC | (SP) | 4 | A,BC,HL saved |
| 09B1H | (HL) | ACC | 4 | HL = HL+ 4 |
| 09B4H | BCDE | ACC | N/A | BC,HL saved |
| 09BFH | ACC | BCDE | N/A | N/A |
| 09C2H | (HL) | BCDE | 4 | HL = HL + 4 |
| 09CBH | ACC | (HL) | 4 | HL = HL + 4 |
| 09CEH | (DE) | (HL) | 4 | HL = HL + 4<br>DE = DE + 4 |
| 09D2H | (HL) | (DE) | NTF | HL = HL + NTF<br>DE = DE + NTF |
| 09D3H | (DE) | (HL) | NTF | HL = HL + NTF<br>DE = DE + NTF |
| 09D6H | (DE) | (HL) | A<br>A= LT 256 | HL = HL + A<br>DE = DE + A |
| 09D7H | (DE) | (HL) | B<br>B= LT 256 | HL = HL + B<br>DE = DE + B |
| 09F4H | AACC | ACC | NTF | N/A |
| 09FCH | ACC | AACC | NTF | N/A |
| 0A9AH | HL | ACC | N/A | N/A |

TABLE 4

--** DATA MOVEMENT **--

(Continued)

* The value is pushed on the stack, a POP BC, POP DE can be used in succession to retrieve the value. In case of an integer, the value will then be in the DE reg. pair.

NOTE: "LT" in table stands for Less Than.

## COMPARE AND TEST ROUTINES

It is often necessary to compare certain numbers or strings with one another or to test a memory location or register to find the type of data it contains. For instance, while searching for a particular number or string in memory. These types of routines can become quite complicated if they have to be written from scratch. This section helps by listing all the compare routines in ROM which can be used.

In Table 5, the column "ADDRESS" is the CALL address, the column headed "FUNCTION" shows the formula for the function performed. Unless stated otherwise, all registers are used however, no memory locations are altered by the compare routines presented here. Remember also, that a compare is a subtraction that is actually never performed as far as the register contents are concerned. Nevertheless the flags are set or reset in the same way as though a subtraction had taken place. It is for this reason that in the FUNCTION column the operation is shown as a subtraction.

The compare routines 1 to 5 set the Z and S flags as though a subtraction had taken place, and load the A reg. with 0 if the two values are equal, with 1 if a subtraction would have resulted in a positive result and with FFH if a subtraction would have resulted in a negative result. Note that using compare 1 with a NTF of 3 will result in an error. Then there is the RST 8 and the RST 10H. These are used for scanning strings.

RST 10H does the following:

```
1D78   23      INC   HL
1D79   7E      LD    A,(HL)     ;GET FIRST CHARACTER
1D7A   FE3A    CP    3AH        ;IS CHARACTER NUMERIC?
```

```
1D7C    D0       RET    NC         ;RETURN IF IT IS NOT
1D7D    FE20     CP     20H        ;IS CHAR ASCII SPACE ?
1D7F    CA781D   JP     Z,1D78H    ;LOOP TO BYPASS SPACE
1D82    FE0B     CP     0BH        ;IS IT GREATER THAN 0AH
1D84    3005     JR     NC,1D8BH   ;GO FIX C FLAG.
1D86    FE09     CP     9H         ; IS IT CONTROL CHAR ?
1D88    D2781D   JP     NC,1D78H   ;LOOP TO BYPASS
1D8B    FE30     CP     30H        ;IS CHAR NO 0 TO 9 ?
1D8D    3F       CCF               ;C FLAG SET IF NUMERIC
1D8E    3C       INC    A          ;ELSE RESET C FLAG.
1D8F    3D       DEC    A          ;SET Z FLAG IF A = 00H
1D90    C9       RET               ;RETURN WITH FLAGS SET.
```

This routine bypasses spaces and control characters (vertical and horizontal tabs) and sets the C flag if the character is numeric. The HL reg. will point to the first non-blank character in the string.

--** RST 8 **--

The RST 8 routine compares the character pointed to by HL with the value pointed to by the two bytes on the top of the stack (which is the return address in this case). Care must be taken here because, if they are unequal, a SN ERROR will be generated. If they are equal, the return address will be incremented to bypass the test character (which should be placed right after the RST 8 in your programs). Before this routine returns it will call RST 10H to find the next non-blank character. This routine can be used to check for certain characters and point to the next non-blank character after execution. It must also be noted that disk system users should CALL the actual addresses listed in the ADDRESS column and not use the RST 8H or RST 10H instructions.

TABLE 5

--** COMPARE AND TEST ROUTINES **--

| NO. | ADDRESS | FUNCTION | ROUTINE TYPE (NTF) |
|---|---|---|---|
| 1 | 0994H | SGN (ACC) | * NTF=2, 4 or 8, ERROR if NTF=3, all registers saved |
| 2 | 0A0CH | ACC-BCDE | * NTF=4 only, all registers saved |
| 3 | 0A39H | HL-DE | * Twos complement, all registers saved |
| 4 | 0A4FH | ACC-AACC | * NTF = 8 |
| 5 | 0A78H | AACC-ACC | * NTF = 8 |
| 6 | RST18<br>1C90H | HL-DE | BC, DE and HL saved, unsigned compare. |
| 7 | RST 8H<br>1C96H | (HL)-((SP)) | BC,DE saved |
| 8 | RST 10H<br>1D78H | (HL) | BC, DE saved, C and Z flags used HL, incremented see text. |
| 9 | RST 20H<br>25D9H | NTF | A = NTF - 3, Z & S flags valid, C set if NTF not equal to 8, C is reset if NTF = 8, Z set if NTF = 3 |

TABLE 5

--** COMPARE AND TEST ROUTINES **--

(Continued)

| NO. | ADDRESS | FUNCTION | ROUTINE TYPE (NTF) |
|---|---|---|---|
| | --** The following routine checks the ACC **-- | | |
| 10 | 0955H | ACC | If ACC=0, Z flag is set |
| | --** This routine checks for strings **-- | | |
| 11 | 0AF4H | NTF | If NTF not = 3, TM ERROR, BC,DE and HL saved |
| --** Checks the char. HL is pointing to for type **-- | | | |
| 12 | 1E3DH | (HL) | C flag set if (HL) points to ASCII letter value or rest otherwise. |

* A reg. = 0 if equal
A reg. = 1 if greater than 0
A reg. = FFH if less than 0

Note: Compare routines 1 to 6 above also set or reset the S and Z flags.

- LEVEL II ROM REFERENCE SECTION -

## --** DATA CONVERSION ROUTINES **--

So far, we have discussed routines for arithmetic and compare functions. Some functions, however, will only work with one type of data format. Additionally, input from the keyboard needs to be converted from ASCII to Hexadecimal before it can be used, while the opposite process is necessary when printing a number to the printer or video display. This section describes ROM routines which perform such functions.

....oooo0000oooo....

TABLE 6.

--** CONVERSION LOGIC **--

| ADDRESS | INPUT | OUTPUT | COMMENTS |
|---|---|---|---|
| 0E65H | (HL) | ACC,NTF=8 | This routine converts an ASCII string pointed to by HL into a number stored in the ACC. The string must end with a comma or ZERO byte. (AACC is changed). After execution HL will point to the last byte of the string. (0H or ,) |
| 0E6CH | (HL) | ACC | This routine performs the same function as E65H above except that the output in the ACC is of the smallest NTF type possible for the size of the number. |
| 0FBDH | ACC | (HL) | This routine converts a number contained in the ACC to an ASCII string. The start address of the string will be returned in the HL register pair. The string is terminated with a ZERO byte. (ACC and AACC are changed). |
| 1E5AH | (HL) | DE | This routine converts a numerical ASCII string pointed to by the HL register pair to a BINARY value. The result is |

TABLE 6. CONTINUED.

--** CONVERSION LOGIC **--

| ADDRESS | INPUT | OUTPUT | COMMENTS |
|---|---|---|---|
| | | | placed in the DE register pair. For more information see PART 2. |
| 2BC1H | (HL) | A | The main function of this routine is to convert a numerical ASCII string pointed to by HL to BINARY. The result is placed in the A register. For more information see PART 2. |

- LEVEL II ROM REFERENCE SECTION -

--** INPUT ROUTINES **--

These routines are used to input data from the keyboard. There are two different types of input routines. The first type returns with one character only, while the other allows a string of characters to be input into an input buffer. At this stage, it is appropriate to mention that the SYSTEM command sets the stack pointer to 4288H which is right in the middle of the input buffer normally used by BASIC. To overcome this, set either the stack pointer or the buffer to a different location in memory. The buffer can be relocated by putting a suitable address in memory location 40A7H which is the I/O buffer pointer.

The 361H and 1BB3H routines pass control to 41AFH which contains a RET (C9H) instruction if Level II is used. Disk BASIC however, uses this location to jump to DOS code which could cause unreliable results. If these routines are called while disk BASIC is enabled, it is best to set 41AFH to C9H.

TABLE 7.

--** CHARACTER INPUT ROUTINES. **--

| ADDRESS | OUTPUT | COMMENTS |
|---|---|---|
| 002BH | REG. A | This routine scans the keyboard and returns with the ASCII value of the key pressed in the A register. If no key was pressed A will contain 0H. BC and HL are saved. |
| 0049H | REG. A | This routine is the same as 2BH above, except that it will not return until a key is pressed. BC and HL are saved. |
| 0050H | REG. A | (Model 3 only) This routine will input a character from the RS-232 interface. It will either wait for a character or return if there is no character depending on how the "WAIT" switch was set on initialization (see 005AH). BC and HL are saved. |
| 035BH | REG. A | This routine simulates the INKEY$ function. The rules for this routine are the same as for the 2BH routine. The ONLY difference is that ALL registers are saved whereas the 2BH routine destroys the contents of the DE register pair. |

TABLE 8.

--** STRING INPUT ROUTINES **--

| ADDRESS | COMMENTS |
|---|---|
| 0040H | This location jumps to 5D9H therefore the same rules apply. |
| 0361H | This routine inputs a string from the keyboard and stores it at the buffer location pointed to by the I/O buffer pointer 40A7H. The string may be up to 240 bytes long. The input string will be terminated with a ZERO byte automatically, as soon as the ENTER key is pressed. After execution, the HL register pair will point to one byte before the start of the string so that RST 10H can be used to locate the first non-blank character. |
| 05D9H | This routine is the most basic and the most flexible of the INPUT routines. The HL register pair must point to the first byte of the input buffer, the B register must be loaded with the maximum buffer length required. Keyboard input over the maximum buffer length is ignored. After pressing the ENTER key it will return with HL containing the original buffer address and B will contain the actual input string length. |
| 1BB3H | The function of this routine is identical to that of the 361H routine with the exception that it prints a question mark, before input is accepted. |

## --** STRING OUTPUT ROUTINES **--

There are several output routines available in ROM. Some are general purpose routines which will output to printer, video or tape. Others will only output to the display. The general purpose routines invariably use the location 409CH as a flag byte to indicate the device to which the output is to be directed. If this location is loaded with 00H the output will go to the video display. If it is loaded with 01H output will be directed to the line printer, and if it is loaded with -1 (FFH) the output goes to tape.

The routine at 2170H can be used to set the device type flag at 409CH to zero for video output. In level 2 mode all registers are saved after a CALL 2170H.

TABLE 9.

--** SINGLE CHARACTER OUTPUT ROUTINES **--

| ADDRESS | | COMMENTS |
|---|---|---|
| 0033H | | This routine will output a byte from the A register to the video display. BC and HL saved. |
| 003BH | | This is similar to 33H except that output goes to the line printer. BC and HL saved. |
| 0055H | | The character in the A register is output through the RS-232 interface. (DE is used). |
| 032AH | * | This routine will output a byte from the A register to the display, printer or tape depending on the state of the DEVICE TYPE FLAG at 409CH. (see Part 2) |
| 033AH | | This routine performs the same function as 33H, the ONLY difference is that the DE register pair is saved. This means that ALL the GENERAL PURPOSE registers are saved. |
| 039CH | | This routine outputs a character from the A register to the line printer, it is the same as 3BH except that the DE register pair is saved. |

* These routines are able to output data to tape ONLY if the correct procedure is followed for setting up the cassette. See the tape section for more detail.

TABLE 10

--** STRING OUTPUT ROUTINES **--

| ADDRESS | | COMMENTS |
|---|---|---|
| 021BH | | (Model 3 only) This routine will display a line (string) on the crt. The string must be terminated with a byte 03 or 0DH, 03 will leave the cursor at the next character position after the last character printed, and 0DH (CR) will scroll the cursor to the start of the next line. HL must point to the string on entry, on exit HL will point the the byte following the terminator. (DE is used). |
| 28A7H | * | This routine will output a string of data to the display, printer or tape depending on the contents of 409CH. The HL register pair must point to the start of the string to be output. The string must be terminated with a zero byte (00H) or a quote (22H). |
| 2F0AH | * | This routine will output a string of data to the display, printer or tape, again depending on 409CH. The difference between 28A7H above and this routine is that the D and B registers play a role. The D register should be loaded with the maximum number of characters which should be printed. The routine will stop when either the maximum number of characters has been printed or if the delimiter is reached, which should be a ZERO byte. If the B register is set to zero before this routine is called the B register will reflect the actual number of characters |

TABLE 10

--** STRING OUTPUT ROUTINES **--

(Continued)

printed. Otherwise, it will simply be incremented by this amount.

* These routines are able to output data to tape ONLY if the correct procedure is followed for setting up the cassette. See the tape section for more detail.

## --** DEMONSTRATION PROGRAM **--

The program listed below demonstrates input, output and arithmetic routines. When this program is executed it will print a question mark, if this is answered with two numbers separated with a comma then they will be multiplied and the result printed on the screen.

```
010          ORG       7000H
020  START   CALL      1BB3H     ;INPUT X,Y
030          RST       10H       ;FIND FIRST CHAR.
040          CALL      0E6CH     ;PUT X VALUE IN ACC
050          PUSH      HL
060          CALL      0AB1H     ;CSNG
070          CALL      09BFH     ;STORE IN BCDE
075                                ;REGISTERS
080          EXX                 ;SAVE REGISTERS
090          POP       HL
100          RST       8H        ;CHECK FOR COMMA
110          DEFB      ","
120          CALL      0E6CH     ;PUT Y VALUE IN ACC
130          CALL      0AB1H     ;CSNG
140          EXX                 ;RESTORE REGISTERS
150          CALL      0847H     ;MULTIPLY X AND Y
160          LD        HL,BUFR
170          CALL      0FBDH     ;PUT RESULT IN BUFR
180          CALL      2170H     ;SET DTF FOR VIDEO
190          CALL      28A7H     ;PRINT RESULT
200          LD        A,0DH
220          CALL      33H       ;PRINT CR.
230          JR        START     ;DO IT AGAIN
245  BUFR    DEFS      240       ;240 BYTE BUFFER
250          END       START
```

## --** TAPE I/O ROUTINES **--

Using tape I/O in assembly language programs is straight forward. All tape I/O can be handled by a series of CALLs. Before using the tape I/O routines it is a good idea to read the data format section to find out how data is actually stored on tape. Also note that the programmer will have to determine when to stop reading from tape. The data format section can be consulted, to find the appropriate end-of-file pointers.

* The general purpose output routines 32AH and 28A7H can also be used to output to tape, however 212H and 287H will have to be used to define the drive and to write the leader as normal. Note that in the Model 3 the routine at 212H has been removed. All that 212H does in the Model 3 is clear the A register and return. This means that programs written for the Model 1 will still work. The routines for reading and writing the tape leader will automatically turn the cassette recorder on, so the Model 3 does not have to use the routine at 212H. For compatability, software authors, should CALL 212H regardless of the computer system being used.

## -- DEMO TAPE I/O ROUTINE --

This is a tape I/O routine which can either read or write 255 bytes, depending on lines 60 and 70. A CALL 264H in line 70 will write to tape and a CALL 235H will read from tape.

```
10          ORG         7000H
20   START  XOR         A           ;CLEAR A
30          CALL        212H        ;DEFINE DRIVE 1
40          LD          HL,5000H    ;START ADDRESS
50          LD          B,0FFH      ;INIT LOOP COUNTER
60   LOOP   LD          A,(HL)      ;GET BYTE
70          INC         HL          ;POINT TO NEXT BYTE
80          CALL        264H        ;WRITE IT TO TAPE
90          DJNZ        LOOP        ;LOOP 255 TIMES
100         CALL        1F8H        ;TURN CASSETTE OFF
110         JP          6CCH        ;BACK TO BASIC
120         END         START
```

The read routine would be identical except for lines 60 and 80 and line 65 must be inserted.

```
60   LOOP   CALL        235H        ;READ FROM TAPE
65          LD          (HL),A      ;PUT IT IN MEMORY
```

Line 80 must be deleted for the read routine.

TABLE 11

--** TAPE I/O AND CONTROL **--

| ADDRESS | COMMENTS |
|---|---|
| 01F8H | This routine simply turns the tape recorder off. BC, DE and HL are unchanged. |
| 0212H | This routine is used to turn the tape recorder on. The A register should contain 00H to turn recorder 1 on or FFH to turn recorder 2 on. BC, DE and HL are saved. (N/A to Model 3) |
| 022CH | This routine will blink the asterisk in the right hand top corner. AF is destroyed. Note that this routine will only function properly if location 3C3FH contains an ASCII blank or an asterisk. If necessary, a CALL 29FH can be used to turn both the asterisk on when required. |
| 0235H | This routine will read a byte from tape and return with it in the A register BC, DE and HL are saved. |
| 0264H | Writes a byte from the A register to tape. BC, DE and HL are saved. |
| 0287H | Writes tape leader and A5H sync byte to tape. BC, DE and HL are saved. |
| 0296H | This routine will read until the A5H sync byte has been found. It will bypass anything on tape until the sync byte is located including the leader. BC, DE and HL are saved. |

TABLE 11

--** TAPE I/O AND CONTROL **--

(Continued)

| ADDRESS | COMMENTS |
|---|---|
| 029FH | This routine is the part of the routine at 0296H which places the two asterisks in the top right corner of the screen when the sync byte is found. 029FH may be called independently of 0296H. (Model 1 only) |
| 0314H | This routine reads two bytes and returns with them in the HL register pair. |

## -VARIABLE ORGANISATION AND VARIABLE LOCATING ROUTINES-

This section deals with routines which are used to locate variables created by BASIC and one routine which can interpret a BASIC expression and store the result in the ACC. The routines located at 260DH and 2540H allow the passing of values from BASIC to a machine language subroutine. The number of values that can be passed is dependent only on the number of variables allowed by BASIC.

These routines are quite simple to use. To use routine 260DH for example, simply make the HL register pair point to the first character of an ASCII string representing the name of the variable whose value is required, then execute a CALL 260DH. After execution, the DE register pair will contain the address of the variable in memory. This enables you to locate the value in memory and operate on it. Note that if the variable does not exist it will be created and given a value of zero. Therefore, care must be taken to use variables already created, because creation of a new variable could mean that all other variables are moved in memory and addresses already returned by the 260DH routine will no longer be valid.

With string variables the address returned in the DE register pair does not point to the variable directly but to the first of three bytes containing the string length followed by the actual string address. The 2540H routine performs a similar function to 260DH except that the value of the variable is placed in the ACC and the NTF is set accordingly. For string variables this routine will load the ACC with three bytes containing the length and address of the string.

Finally, we come to 2337H which is a very useful routine. It allows the user to execute BASIC expressions during a machine language subroutine.

Input to this routine consists of a string containing a BASIC expression terminated (delimited) by a colon, comma, right bracket ")" or a zero byte. The HL register pair should be made to point to the first character of the expression. After execution of a CALL 2337H the result will be in the ACC and the NTF will be set appropriately. In the case of strings the ACC will contain the three bytes indicating string length and string location address. It is important to note that this routine makes considerable use of the stack and also the machine must be in the RUN mode for it to work as expected. All routines presented here will return with the HL register pair pointing to the delimiter.

TABLE 12.

--** SPECIAL PURPOSE ROUTINES **--

| ADDRESS | | INPUT | OUTPUT |
|---|---|---|---|
| 2540H | * | (HL) = First char. of ASCII var. name | ACC, NTF, HL points to delimiter |
| 260DH | * | (HL) = First char. of ASCII var. name | DE = Start address of value of variable |
| 2337H | * | (HL) = First char. of ASCII string delimited with :,) or ZERO byte. | ACC, NTF |
| 29D7H | | This routine will make the HL register pair point to the data in the ACC. NTF must be set before CALLing this routine. | |

* See text.

## --** VARIABLE ORGANIZATION **--

This section explains the format used for variables by BASIC. NUmeric variables are stored in memory as follows:

NTF - This is the number type. (also length of data)
CHR2 - Second character of variable name.
CHR1 - First character of variable name.
DATA - The data will be stored in the same format as shown in Table 1.

String variables are somewhat different. The data following the variable name consists of three bytes, the first of which is the actual string length while the second and third form the actual string location address.

Finally, there are array variables. The data element for these is different again. The first two bytes after the variable name contain the size of the array (i.e. the number of bytes used). The third byte contains the number of dimensions used, next there follow two bytes for each dimension in the array which indicate the number of data elements in each. (As this includes the zero element, the values in these bytes are always one higher than the original DIMensioned size). The data is arranged so that the first index varies the fastest. In other words, if an array is DIMensioned to be: DIM A(2,2), then the data will be stored in the following sequence: 0,0 1,0 2,0 0,1 1,1 2,1 0,2 1,2 2,2

## --** ERROR ROUTINES **--

When writing machine language subroutines it may be necessary at times to test for errors. After finding an error it is often tedious to let the outside world know that an error has occurred, not to speak of the memory wasted by storing error messages. To overcome this, BASIC error messages can be used. It is for this purpose that a variety of error locations are given in TABLE 13. To use these just jump to the address given. The error message will be printed and control will be passed back to the BASIC command mode.

Which brings up another related subject. Whenever an error occurs while in the BASIC mode, a CALL is made to location 41A6H, before the error message is printed to the display. Normally this location contains a RETurn instruction (C9H) which means that it will return to whence it came, straight away. However this location can be used to pass control to a machine language routine used for error trapping etc. As a matter of fact Disk Basic uses this location to pass control to a routine that prints the error message in full instead of the abbreviated form that Level II uses.

TABLE 13

--** ERROR ROUTINES **--

| ADDRESS | ERROR MESSAGE TYPE. | |
|---|---|---|
| 012DH | L3 ERROR | (DISK BASIC ONLY) |
| 07B2H | OV ERROR | (OVER FLOW) |
| 0AF6H | TM ERROR | (TYPE MISMATCH) |
| 197AH | OM ERROR | (OUT OF MEMORY) |
| 1997H | SN ERROR | (SYNTAX ERROR) |
| 199AH | /0 ERROR | (DIVIDE BY 0 ERROR) |
| 199DH | NF ERROR | (NEXT WITHOUT FOR) |
| 19A0H | RW ERROR | (RESUME WITHOUT ERROR) |
| 1E4AH | FC ERROR | (ILLEGAL FUNCTION CALL) |
| 1EECH | RG ERROR | (RETURN WITHOUT GOSUB) |
| 2003H | UE ERROR | (UNDEFINED ERROR) |
| 27EDH | BS ERROR | (BAD SUBSCRIPT) |
| 2831H | ID ERROR | (ILLEGAL DIRECT) |
| 28A1H | ST ERROR | (STRING TOO COMPLEX) |

## --** VIDEO CONTROL **--

This section handles a variety of routines relating to control of the video display. There is a routine to clear the screen completely, and there is a routine to clear from a predetermined position to the bottom of the screen. Then there are the routines to change from 64 to 32 characters per line and vice versa. The only routine that needs further explanation is the clear-to-end-of-frame routine. To use this routine, the HL register pair must be loaded with the location from which you wish to start erasing.

Often it is necessary to clear the screen from the cursor location onwards. To do this simply load the HL register pair with the contents of locations 4020H and 4021H, which contain the cursor position and CALL 57CH. (5C5H for Model 3)

TABLE 14.

--** VIDEO CONTROL ROUTINES **--

| ADDRESS | INPUT | COMMENTS. |
|---|---|---|
| 01C9H | N/A | Performs the CLS function |
| 01D9H | N/A | (Model 3 only) This routine dumps the screen contents to the line printer, graphics characters are printed as periods (ASCII 2EH). |
| 01DCH | (HL) | (Model 3 only) The screen contents starting at the location in screen RAM to which HL points, are dumped to the printer. Graphics are handled in the same way as the routine above. |
| 04C3H ** | N/A | This routine will change the display to 64 characters per line. |
| 04F6H ** | N/A | This routine will change the display to 32 characters per line. |
| 057CH (5C5H)* | | HL (see text) Clear screen to end of frame routine. (starts at location (HL). |

* - Addresses in brackets are for the Model 3.
**- See section on System mask byte (4210H) for these functions.

## --** GRAPHICS **--

The graphics routines are somewhat difficult to use. This is both because they are part of the BASIC decoding logic and because a RST 8H is done at the end of each routine. A dummy string can be used however, in order to satisfy the RST 8H logic. Let us assume that it is necessary to SET a graphic block at location X=65 and Y=23 which is about in the middle of the screen. In its simplest form the program would look like this:

```
100             ORG       5000H
110             LD        B,65        ;LOAD X COORDINATE
120             LD        A,23        ;LOAD Y COORDINATE
130             LD        H,80H       ;LOAD "SET" FLAG
140             CALL      GRAFIX      ;PUSH RET. ADDRESS
145                                   ;ON STACK
150             HALT                  ;ALL DONE.
160  GRAFIX     PUSH      HL          ;PUSH FLAG
170             PUSH      BC          ;PUSH X COORDINATE
180             LD        HL,188CH    ;POINT HL TO DUMMY
185                                   ;STRING. ")+"
190             JP        150H        ;JUMP TO GRAPHICS
195                                   ;LOGIC.
200             END
```

Now, let us go through this one line at a time. The program starts with an ORG statement to tell the assembler to start at 5000H. Next, line 110 loads the B register with the X coordinate and line 120 loads the A register with the Y coordinate. Line 130 loads the H register with 80H. 80H is the flag for the SET function. The graphics routine can perform the POINT, SET and RESET functions, depending on the flag byte

--** GRAPHICS **--

(Continued)

passed to it in the H register. The following flag values can be used:

80H = SET
01H = RESET
00H = POINT

The POINT logic will return with 0 in the ACC if block is SET or with 0FFFFH if block is RESET. Line 140 CALLs the address labelled GRAFIX. The reason we are doing this is to place the RETurn address on the stack. Unless we do this the interpreter will never return control to our subroutine.

Line 150 is at the location which was pushed on the stack by the CALL statement in line 140. When the interpreter has SET the graphics block it will return here. The program will exit at line 150 *. In real applications there would probably be a RET here instead of a HALT, to return to the routine which would have called our graphics routine.

Lines 160 and 170 simply PUSH the coordinates on the stack, this must be done in the order shown. Line 180 loads the HL register pair with the address of a dummy string which happens to be in ROM at location 188CH. It is not necessary to use the string in ROM but it saves 2 bytes. The dummy string is needed to satisfy the RST 8H logic which will be looking for a closing bracket ")".

At this point we are ready to enter the graphics logic at 150H. All we have done so far is load registers and push values on the stack. The reason being that when we enter the interpreter we must fool

--** GRAPHICS **--

(Continued)

it in to thinking that it is executing a BASIC statement. In other words, we must simulate the condition the registers and the stack would be in if we were in BASIC and bout to execute a graphics command. As with most BASIC functions the RST 8H routine is used to look for delimiters at the end of the function. The graphics functions SET, RESET and POINT all have a closing bracket at the end of them, hence the need for a dummy string somewhere in memory, consisting of a bracket ")" followed by a non zero byte.

After reading this section carefully there should be no major problems using graphics. Of course, if a line has to be SET, then this sort of routine need have to be put in some form of loop.

* Normally the Z80 CPU stops execution when a HALT opcode is encountered. However the TRS-80 Model 1 is configured to cause a NMI when a HALT is encountered. So a HALT has the same effect as pushing the RESET button.

## --** KEYBOARD MATRIX **--

The keyboard is a matrix of switches which are decoded as though they are part of the memory. The keyboard area is located from 3800H to 3BFFH. If the format of this matrix is known, it is quite simple to use the keyboard for input directly rather than one of the input routines. Sometimes, it is necessary to scan the keyboard quickly for a particular key (like the BREAK key) and it will be necessary to know where to look. It is also useful to know how BASIC scans the keyboard. Each keyboard address is read in succession. If a non-zero value if found at any of these addresses a key has been pressed. Each keyboard address corresponds to a row of the keyboard, each row in turn has a corresponding buffer location where the previous byte read is stored. This is used for the keyboard rollover feature. If a non-zero value is read from a row it is exclusive or-ed with the value in the corresponding buffer location. This means that if the key was pressed the last time the keyboard was scanned it will be ignored. If the keyboard buffer from 4036H to 403CH is set to zero at regular intervals the keys will actually repeat as long as they are depressed.

The address of the keyboard rows is shown in the following table, on the left. The corresponding buffer location for each row is on the right. The numbers in brackets under the bit numbers are the values with which to AND to find if a particular key is pressed. For example, if location 3804H is read and the "T" key is depressed, bit 4 would be set. To test if bit 4 is set we can AND it with 10H. If, after ANDing with 10H, there is a NON-zero value left then the "T" key was pressed. A small demo program is shown after the table.

- LEVEL II ROM REFERENCE SECTION -

--** KEYBOARD MATRIX **--

| BIT NO. | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | BUFFER |
|---|---|---|---|---|---|---|---|---|---|
| ADDRESS | (80) | (40) | (20) | (10) | (08) | (04) | (02) | (01) | LOCATION |
| 3801H | G | F | E | D | C | B | A | @ | 4036H |
| 3802H | O | N | M | L | K | J | I | H | 4037H |
| 3804H | W | V | U | T | S | R | Q | P | 4038H |
| 3808H | | | | | | Z | Y | X | 4039H |
| 3810H | SQ | & | % | $ | # | " | ! | | |
| | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | 403AH |
| 3820H | ? | LT | = | GT | + | * | ) | ( | |
| | / | . | - | , | ; | : | 9 | 8 | 403BH |
| 3840H | SPC | RA | LA | DA | UA | BRK | CLS | ENT | 403CH |
| 3880H | | | | | | | | SHFT | |

LT = Less than symbol.
SPC = Space.
LA = Left arrow. (Backsp)
UA = Up arrow. (Esc)
CLS = Clear key.
SHFT= Shift key.

GT = Greater than symbol.
RA = Right arrow.
DA = Down arrow. (Ctrl)
BRK = Break key.
ENT = Enter key. (Newline)
SQ = Single quote.

## --** KEYBOARD MATRIX **--

(Continued)

The following program further demonstrates keyboard scanning. The routine checks the keyboard for the BREAK key.

```
100     LD          A,(3840H)     ;READ FROM ROW
110                               ;CONTAINING THE BREAK
120                               ;KEY.
130     AND         04H           ;MASK AND SET FLAGS
140     JP          NZ,....       ;JP IF BREAK KEY IS
150                               ;PRESSED.
160     ....                      ;WILL CONT. HERE IF NOT
```

## --** DOS LINK ADDRESSES **--

The level 2 BASIC interpreter was written with upward expandability in mind. Right from the start it was decided that there would be a Disk BASIC version with more powerful instructions than possible with the Level 2, 12K interpreter. In order to allow for this the Disk BASIC link areas were created. Whenever BASIC finds a Disk BASIC command, it will jump to a unique location in reserved RAM. When Level 2 is in command, these locations contain jumps to the L3 ERROR routine. Disk BASIC loads these locations with the entry points of each individual Disk BASIC routine. The table following shows the link addresses. The entrypoints to the various routines in Disk Basic vary from DOS to DOS and from Version to Version. To get a list of actual entry points it is possible to write a small Basic program to PEEK the link addresses and display the contents.

- LEVEL II ROM REFERENCE SECTION -

--** DISK BASIC LINK ADDRESSES **--

| LINK ADDRESS | COMMAND |
|---|---|
| 4152H | CVI |
| 4155H | FN |
| 4158H | CVS |
| 415BH | DEF |
| 415EH | CVD |
| 4161H | EOF |
| 4164H | LOC |
| 4167H | LOF |
| 416AH | MKI |
| 416DH | MKS |
| 4170H | MKD$ |
| 4173H | CMD |
| 4176H | TIME$ |
| 4179H | OPEN |
| 417CH | FIELD |
| 417FH | GET |
| 4182H | PUT |
| 4185H | CLOSE |
| 4188H | LOAD |
| 418BH | MERGE |
| 418EH | NAME |
| 4191H | KILL |
| 4194H | & |
| 4197H | LSET |
| 419AH | RSET |
| 419DH | INSTR |
| 41A0H | SAVE |
| 41A3H | LINE |

Level 2 users can use these addresses to make the machine jump to a machine language program.

## --** INTERCEPT ADDRESSES **--

There are some other addresses worth mentioning. First, there is 41A6H. As mentioned before, this address can be used to intercept and trap errors. Then there is 41BBH, this address can be used to intercept the initialization routine. 400CH can be used to intercept the BREAK key routine. 41C4H is also a very useful interface routine, BASIC always jumps to this address before executing a line. If this link address is used, HL will be pointing to the start of the BASIC program line to be processed next. The keyboard scanning routine can be intercepted by loading location 4015H with ZERO (THIS WORKS ON MODEL 1 ONLY) and putting a jump at 4033H to the entry point of your machine language routine. This method can be used to intercept characters before BASIC can respond to them. The same effect can be achieved by loading the keyboard driver address at 4016H-4017H with the entry point of your routine directly. The only difference is that location 4015H can be used as a convenient toggle to direct keyboard control to the user and back to the computer whenever it is desired to do so. In either case, the first thing that must be done when your routine takes control is to CALL the keyboard scanning routine at 03E3H (Model 3 use 3024H). For example, assume that we want to disable the BREAK key. The code that the break key returns is 01H. A routine to disable it looks like this:

## --** INTERCEPT ADDRESSES **--

(Continued)

```
10              ORG       4016H
20    DEFW      START
60              ORG       7000H
70    START     CALL      03E3H       ;(3024H Model. 3)
                                      Scan keyboard.
80              CP        01H         ;Is it BREAK key?
90              RET       NZ          ;Carry on if not.
100             XOR       A           ;Clear A to ignore
110             RET                   ;BREAK key.
120             END
```

CALLing 03E3H (Model 3 use 3024H) will result in the A register being loaded with the ASCII code of whichever key is being pressed at the time. It is therefore a simple matter to compare if the character returned is the one you are looking for.

## --** MISCELLANEOUS **--

This section handles a mixture of routines which do not fall under any particular heading. The routines are handled according to their location in ROM.

-- TABLE 15. --

| ADDRESS | COMMENTS |
|---|---|
| 005AH | (Model 3 only) This is the routine that initializes the RS-232 interface. (See Model 3 manual). |
| 0069H | (Model 3 only) This routine initializes all I/O vectors, this routine can be used to undo a device ROUTE. |
| 006CH | (Model 3 only) The I/O device routing can be changed by using this routine. Load 4222H with code for source device, and 4220H with code for destination device. (Codes are two character ASCII : KI,DO,RI,RO,PR) See part 2 for more detail. |
| 0060H | The delay loop is located here. This routine uses the BC register pair as a loop counter and loops until BC is decremented to zero. The time delay is the value in the BC register pair multiplied by 14.65 micro seconds approx. The A register is used. Model 3 users can use this routine to the same effect, the clock in the Model 3 runs faster but the routine was altered to give the same delay. |

-- TABLE 15. --

(Continued)

| ADDRESS | COMMENTS |
|---|---|
| 00EFH | If the HL register pair is loaded with the desired memory size a JUMP to this location will set all required pointers for memory size, stack and variables. |
| 028DH | (Model 3 only) This routine scans the keyboard and looks for the break key. If the Z flag is NOT set then BREAK key is pressed. (A is used) |
| 0298H | (Model 3 only) A call to this routine will turn the CLOCK on. All registers saved except A. |
| 02A1H | (Model 3 only) Turns the clock off. A register used only. |
| 02B5H | A jump to this location will pass control to the SYSTEM routine. This might be useful at times if it is necessary to load a system tape to memory. |
| 06CCH | This is a good location to re-enter the BASIC interpreter from a machine language program (Model 3 can only use 1A19H). |
| 1A19H | This is a re-entry point for BASIC as well. However 6CCH is recommended for the Model 1 instead as 1A19H can cause error mesages such as OM error at times. Model 3 users can only use 1A19H as 6CCH is removed. |

-- TABLE 15. --

(Continued)

| ADDRESS | COMMENTS |
|---|---|
| 1A76H | This is the start of the interpreter command mode. It prints the BASIC prompt, and then scans the keyboard for BASIC commands. |
| 1AF8H | This routine is very useful. It will check and repair the line pointers (if necessary) in a BASIC program. This function is needed after shifting or relocating lines of BASIC program in memory, as the line pointers would otherwise be invalid. |
| 1B2CH | This routine will search a BASIC program for the location of a BASIC statement line, the number of which corresponds to the value in the DE register pair. If a match is found, the carry flag will be set and the BC register pair will point to the start of the line in question. HL will then point to the next line. |
| 1B4DH | A NEW without clearing the screen can be done from a machine language routine by calling 1B4DH. This routine can also be used to reset the BASIC program pointers to any location in BASIC RAM. To do this load the start of BASIC pointer (40A4H) with the new start address and CALL 1B4DH to reset all the pointers. |

-- TABLE 15. --

(Continued)

ADDRESS — COMMENTS

1B5DH — From here you can RUN a BASIC program. To do this use the following procedure:

```
LD      HL,1D1EH
PUSH    HL
JP      1B5DH
```

1B6EH — A CALL to this location will reset most BASIC pointers in reserved RAM.

21C9H — INPUT routine can be CALLed from here. To use, load HL with string address and A with the first character. For example to execute INPUT "AMOUNT";X use the following procedure:

```
INPUT  DEFM     ""AMOUNT";X"
       DEFB     0
       LD       HL,INPUT
       LD       A,(HL)
       CALL     21C9H
```

The only other requirement is that the variable (X in this case) must already exist.

3033H — (Model 3 only) This routine will return the date. To use this routine load HL with an 8 byte buffer address. On exit HL will contain the start address of the buffer and the buffer will contain the date in the folowing format: MM/DD/YY.

-- TABLE 15. --

(Continued)

| ADDRESS | COMMENTS |
|---|---|
| 3036H | (Model 3 only) This routine functions like the routine at 3033H except that it returns the time in the 8 byte buffer. Format: HH:MM:SS. |

## --** DATA AND TAPE FORMATS **--

This section reveals the format of BASIC and MACHINE LANGUAGE programs and data files, both in memory and on tape. The bytes marked with an asterisk (*) are on tape only and not in memory. This means that the files are stored on tape in the same manner as they appear in memory, with the exception of a leader and some bytes at the beginning of the file or block.

### -- BASIC PROGRAM FORMAT --

| | | |
|---|---|---|
| * | LEADER | A leader consisting of 256 zero bytes is found at the start of each file. |
| * | 0A5H | 0A5H is the SYNC byte. |
| * | D3 D3 D3 | BASIC header code. This shows it is a BASIC program. |
| * | NAME | A one character file name is found here. |
| | LSB | LINE POINTER. Points to start of NEXT program line. |
| | MSB | |
| | LSB | LINE NUMBER. The line number is stored here in binary. |
| | MSB | |
| | . | |
| | . | Actual program line up to 240 bytes long is stored here. |
| | . | |
| | 00H | ZERO BYTE. A zero byte signifies the end of line. |

## -- BASIC PROGRAM FORMAT --

(Continued)

A NEW LINE will start here starting with the next line pointer -- OR -- the end of program will be marked here with two zero bytes

00 00

## -- SYSTEM TAPE FORMAT --

| | | |
|---|---|---|
| * | LEADER | |
| * | 0A5H | SYNC byte. |
| * | 055H | Header code for Machine Language programs. |
| * | NAME | Six byte long ASCII file name. If name is less than six bytes long it will be padded with blanks. Although the file name is six bytes long the SYSTEM command only looks at the first character. |
| * | 03CH | Block header code. |
| * | XX | Block length from 1 to 256 bytes. (0 = 256) |
| * | LSB | Starting location of this block in memory. |
| * | MSB | |
| | . | Block of data located here. |
| | . | |
| * | XX | Checksum. The total of starting address and all data in this block added, with any carry ignored. |
| | | The block from 3CH onwards is repeated until the program is complete. |
| * | 078H | END of file code. |
| * | LSB | ENTRY POINT of program is stored here. |
| * | MSB | |

## - FORMAT OF A SOURCE FILE FROM EDTASM -

| | | |
|---|---|---|
| * | LEADER | |
| * | 0A5H | SYNC byte. |
| * | 0D3H | Start code. |
| * | NAME | Six byte long name. Names shorter that six characters are padded with blanks. |
| | LINE NO. | The line number will be stored in ASCII and is five bytes long. Note that these five digits have the most significant bit set. |
| | 20H | Blank spacer. |
| | . | |
| | . | ASCII coded source code stored here. |
| | . | |
| | 0DH | Cariage return marks end of line. |
| | | Text from LINE NO. will be repeated here OR: |
| | 1AH | END of file marker. |

## -- DATA FORMAT FOR FILES CREATED WITH - PRINT # --

| | | |
|---|---|---|
| * | LEADER | |
| * | 0A5H | SYNC byte. |
| | XX | Sign of data. Will be 20H if positive or 2DH if negative. |
| | . | |
| | . | ASCII coded data stored here. |
| | . | |
| | 20H | End of field code |
| | 2CH | Field separator. (ASCII for comma) |
| | | More data here OR: |
| | 0DH | Carriage return marks end of data. |

## -- ADDRESSES USED BY EDTASM --

At this stage it might be worthwhile to give a few addresses which are used by EDTASM. These might come in handy for those times when, after jumping back to BASIC, you realize that you forgot to write the source code to tape. (Model 1 only)

| ADDRESS | USE |
|---|---|
| 4113H | End of memory pointer. |
| 4115H | Start of memory pointer. |
| 41C3H | Start of symbol table pointer. |
| 4301H | Keyboard driver address pointer. |
| 45AAH | Line printer driver. |

## -- INITIALIZING MACHINE LANGUAGE SUBROUTINES --

There are several important features of machine language programs. If the SYSTEM command is used to load and initialize a machine language program, the stack pointer will be set right in the middle of the area used by BASIC as the buffer (4288H). This does not pose any problems for straight machine language programs which do not CALL any input routines in ROM. However, if use is made of such routines, the stack pointer or buffer must be relocated. The buffer can be relocated by loading the buffer pointer (40A7H) with a new address. It is however preferable to move the stack to a different location.

Machine language subroutines used with the USR function do not need any special consideration other than that the stack pointer must contain the same address, when returning, as it did when the subroutine was first entered. There are two options when deciding where to place a machine language routine. It can be placed in high memory, in which case the memory size has to be set by the user to protect it from BASIC. Or a machine language program can be placed in low memory, (4300H onwards for Model 1, 4400H onwards for Model 3) and the BASIC pointers: 40A4H, 40F9H, 40FBH and 40FDH will have to be set a couple of bytes past the end of your machine language program. This should be done immediately after loading the routine to memory.

## -- USING MACHINE LANGUAGE PROGRAMS ON DISK SYSTEMS --

The disk user is faced with some different problems. These are caused by the way the DOS initializes. First let me make clear that there are no problems with straight machine language programs that make no ROM CALLs or with machine language subroutines used from BASIC through the USR function.

The problem stems from the fact that DOS does not initialize the BASIC pointers, jump vectors and link addresses which are used by many of the ROM calls presented in this manual, until BASIC is called up by the user. Most disk users therefore first load the machine language program to memory and then initialize BASIC. After that they use the SYSTEM command to jump to the entry point of their program. This is one way of doing things but it is tedious because the user has to remember the entry point address in decimal for the SYSTEM command. There is however, a better way of initializing your machine language programs.

After some investigation it will be found that the printer, video and keyboard control blocks are initialized by the DOS system on power up, and are untouched by the Disk BASIC initializing procedure when BASIC is called up. This means that the line printer driver address at 4026H - 4027H can be loaded with the entry point to a machine language program. Then, after BASIC is initialized, all that is required is to enter LPRINT and control will be passed to the machine language program. For those users who need to use the line printer all that is required is to reload the proper LPRINT driver address back into the LPRINT driver location during initialization.

## --** PORT 255 **--

Port 255 (0FFH) has several different applications in the TRS-80 Model 1 (see separate section on ports for Model 3). It is also used to set the display to 32 or 64 characters per line. The four least significant bits in the byte sent to this port, control its function. If an OUT (255),A is executed these bits perform the following functions:

| BIT NUMBER | BIT STATUS & FUNCTION |
|---|---|
| 3 | If this bit is set (1) it will cause the video display to display 32 characters per line. If it is reset (0) then 64 characters per line will be displayed. |
| 2 | If this bit is set the cassette recorder will be turned on. If it is reset the recorder will be turned off. |
| 1 and 0 | 00 Writes zero voltage to tape.<br>01 Writes positive voltage to tape.<br>10 Writes negative voltage to tape. |

To see port 255 at work try the small BASIC program below:

```
10 FOR X = 1 TO 4: READ N: OUT 255, N
20 PRINT N: FOR D = 0 TO 500: NEXT
30 NEXT X: STOP
40 DATA 8,4,2,1
50 END
```

## --** THE MODEL 3 SYSTEM I/O BYTE 4210H **--

Unlike the Model 1 the Model 3 uses ports for most of its I/O. The printer, disk controller, cassette switch and RS232 for instance are all port controlled. Other system functions such as 32 and 64 character mode are also port controlled. More often than not a general purpose port is used to control several different things. This means that each bit in the byte output through such a port has an independend function. For instance port ECH (236) has different functions for 5 of its bits. Bit 1 enables or disables the cassette motor and bit 2 selects either 64 character or 32 character mode. Now, if the system wants to turn the cassette motor on for example, it has to know if the screen is in 32 or 64 character mode so that it can output the correct code to port ECH so as not to change the current display mode. In order to overcome this problem BASIC stores the current Port ECH status (plus some more information in bits not used by port ECH as we will see) in a byte in reserved RAM at 4210H. BASIC outputs the value at 4210H to port ECH at regular intervals. So, if you want to turn the cassette relay on for example, it is no good using port ECH directly as BASIC will promptly turn it off again. This applies to all the different functions of port ECH (see the relevant section on Model 3 ports). However, if we change the value at location 4210H (16912), BASIC will output the changed status to port ECH for us and all is well. Of course this only applies if BASIC is in control. If a machine language program is in complete control (including interrupt routines) then port ECH can be used directly.

## --** THE MODEL 3 SYSTEM I/O BYTE 4210H **--

### -- SYSTEM BYTE FUNCTION --

| BIT NO. | DESCRIPTION. | |
|---|---|---|
| | BIT = 1 | BIT = 0 |
| 0 | Clock is on. | Clock is off. |
| 1 | Cassette motor on. | Cassette motor off. |
| 2 | 32 character mode. | 64 character mode. |
| 3 | Alternate character-set enabled. | Alternate character-set disabled. |
| 4 | External (50 way) I/O bus enabled. | External I/O bus disabled. |
| 5 | Video waits enabled. | Video waits disabled. |

## --** MODEL 3 PORT USAGE **--

The following tables are a description of all the ports used by the Model 3.

PORT ADDRESS: E0H
ACCESS MODE: READ
Comment: This port is polled by the interrupt service routine to see which device is interrupting. If a bit is zero it means the device is interrupting.

| BIT NO. | DEVICE. |
|---|---|
| 0 | Cassette 1500 baud rising edge interrupt. |
| 1 | Cassette 1500 baud falling edge interrupt. |
| 2 | Real time clock interrupt. |
| 3 | External I/O bus interrupt. |
| 4 | RS-232 Transmit interrupt. |
| 5 | RS-232 Receive interrupt. |
| 6 | RS-232 Error interrupt. |
| 7 | Undefined. |

PORT ADDRESS: E0H
ACCESS MODE: WRITE
Comment: This port enables or disables the various interrupts. Note that this port has a byte allocated to it at address 4213H. If it is desired to enable or disable some of the interrupts while BASIC is in control, alter the equivalent bit in the byte stored at 4213H. The same principles apply here as for the system byte at 4210H described elsewhere in this manual.

| BIT NO. | When bit = 0 | When bit = 1 |
|---|---|---|
| 0 | Disable cass. rising edge interrupt. | Enable cass. rising edge interrupt. |
| 1 | Disable cass. falling | Enable same. |

--** MODEL 3 PORT USAGE **--

(Continued)

| | | |
|---|---|---|
| | edge interrupt. | |
| 2 | Disable real time clock. | Enable real-time clock. |
| 3 | Disable ext. I/O bus | Enable same. |
| 6 | Disable DISK INTRQ from generating NMI. | Enable same. |
| 7 | Disable DISK DRQ from generating NMI. | Enable same. |
| 4&5 | undefined | |

PORT ADDRESS: E4H
ACCESS MODE: READ

| BIT NO. | When bit = 0 |
|---|---|
| 5 | Reset button is down. |
| 6 | Disk DRQ signal is active. |
| 7 | Disk INTRQ signal is active. |

PORT ADDRESS: ECH
ACCESS MODE: WRITE
Comment: See section on the system mask byte at 4210H.

| BIT NO. | When bit = 0 | When bit = 1 |
|---|---|---|
| 0 | Undefined. | |
| 1 | Cassette motor off. | Cassette motor on. |
| 2 | 64 character mode. | 32 character mode. |
| 3 | Normal character set. | Alternate character set. |
| 4 | Disable ext. I/O bus. | Enable same. |
| 5 | Disable video waits. | Enable same. |
| 6 & 7 | undefined. | |

PORT ADDRESS: F4H
ACCESS MODE: WRITE

| BIT NO. | When bit = 0 | When bit = 1 |
|---|---|---|
| 0 | Select drive 0. | |
| 1 | Select drive 1. | |

--** MODEL 3 PORT USAGE **--

(Continued)

| | | |
|---|---|---|
| 2 | Select drive 2. | |
| 3 | Select drive 3. | |
| 4 | Select side 0. | Select side 1. |
| 5 | No write precomp. | Write precompensation. |
| 6 | No wait states used. | Wait states used. |
| 7 | Single density (FM). | Double density (MFM) |

PORT ADDRESS: FFH
ACCESS MODE: READ
Comment: Reading port FFH clears the 1500 baud cassette interrupts. Bits number 1 upto and including 5 have exactly the same function as the corresponding bits in port ECH.

BIT NO.

| | |
|---|---|
| 0 | 1500 baud cassette bit. |
| 1-5 | See port ECH. |
| 6 | Undefined. |
| 7 | 500 baud cassette bit. |

PORT ADDRESS: FFH
ACCESS MODE: WRITE
Comment: Bits 0 to 1 have the same function as the corresponding bits in port FFH on the Model 1. See the Model 1 port FFH description for these bits.

BIT NO.

| | |
|---|---|
| 0-1 | Cassette output level. (see comment.) |
| 2-7 | Undefined. |

## --** MODEL 3 PORT USAGE **--

(Continued)

| PORT ADDRESS: | -- READ -- | F8H<br>-- WRITE -- |
|---|---|---|
| | Line printer status.<br>BIT 7 = BUSY.<br>BIT 6 = NO PAPER.<br>BIT 5 = SELECT.<br>BIT 4 = FAULT. | ASCII out to printer. |
| ECH | Clear RTC interrupt. | See above. |
| F0H | FDC status register. | Command register (FDC) |
| F1H | FDC track register. | Track register. |
| F2H | FDC sector register. | Sector register. |
| F3H | FDC data register. | Data register. |
| E8H | MODEM status. | UART reset. |
| E9H | N.A. | Load baud rate register. |
| EAH | UART status register. | UART control register. |
| EBH | UART receiver register. | UART transmit register. |

## --** THE SYSTEM 80, VIDEO GENIE, PMC-80 **--

The success of the TRS-80 has encouraged other manufacturers to emulate it. The most widely known TRS-80 compatible computer is manufactured in Hong Kong and distributed throughout the world under a variety of brand names. In Europe this computer is sold as the Video Genie, in America as the PMC-80, in Africa as the TRZ-80 and in Australia as the SYSTEM 80. Throughout this manual, this computer is referred to as the SYSTEM 80.

The Model 1 routines described in this manual are all valid for the System 80 and respond in the same way as in a TRS-80. There are differences in the hardware but these have been taken into account in the relevant ROM routines. However, at times it is useful to access the hardware directly rather than through a ROM call, so for that purpose the hardware differences between the TRS-80 and the System 80 are discussed below.

## --** CASSETTE INTERFACE **--

The cassette interface on both machines uses port FFH as its data port, with bits 0 and 1 being used to output data to the cassette recorder (see discusion on port FFH elsewhere in this manual). However, bit 2 is used differently. The System 80 uses this bit to enable data to reach the recorder. In other words bit 2 must always be held high (bit 2 = 1) when reading or writing data to the cassette. This means that each time port FFH is written to, bit 2 must be 1 if the cassette has to stay enabled. It is also necesary to select which recorder is to be used, internal or external. This is done with port FEH, bit 4. Setting bit 4 to 0 selects cassette 1 (inbuilt) and setting bit 4 to 1 selects cassette 2 (external).

## --** THE SYSTEM 80, VIDEO GENIE, PMC-80 **--

(Continued)

## --** PRINTER INTERFACE **--

The printer port in the System 80 uses port FDH instead of memory address 37E8H. The bit usage of this port is identical to the TRS-80 useage and should not present any problems. To use the printer port, read from port FDH to see if the printer is BUSY (bit 7 will be high if printer is busy). Then output character to port FDH.

### -- SYSTEM 80 CASSETTE PORT SUMMARY --

| PORT FFH. BIT NO. | |
|---|---|
| 0 & 1 | Data output (see TRS-80 port FFH section) |
| 2 | High = cassette on , Low = cassette off |
| 7 | Cassette data in (Read) |

| PORT FEH BIT NO. | |
|---|---|
| 4 | Cassette select: high = No. 2, low = No.1 |

- LEVEL II ROM REFERENCE SECTION -

**** PART 2 ****

- LEVEL II ROM REFERENCE SECTION -

****** LEVEL II ROM MAP ******

| LOCATION | DESCRIPTION |
|---|---|
| 0000-0002 | Disables the interrupts, clears the A register, then jumps to initialization routine at 674H. |
| 0008 | (RST 8H) Jumps to 4000 through 1C96H. This routine is used by BASIC to check for expected delimiters. It compares the character pointed to by the HL register pair with the character pointed to by the return address on the top of the stack (Note that a RST instruction is in effect a CALL and places a return address on the stack) formula: (HL)=((SP))? If they are not equal an SN ERROR will result; if they are equal then the return address on the stack will be incremented to bypass the test character and control will be passed to RST 10H logic. RST 8H can be used to look for expected characters in a string and then return with (HL) pointing to the next non-blank character. (See RST 10H) (BC and DE registers unaffected). This routine can be used by CALLing 1C96H or RST 8H. |

| LOCATION | DESCRIPTION |
|---|---|
| 0010 | (RST 10H) Jumps to 1D78H through 4003H. This routine INCrements HL and tests the characters pointed to by the HL register pair. It will bypass any spaces and ASCII codes 9 and 10 (right arrow (TAB) and down arrow respectively). Upon return from this routine HL will point to the next non-blank character, the carry flag will be SET if HL is pointing to a numeric ASCII character and the Z flag will be SET if the character pointed to happens to be zero or 3AH (":"). (BC and DE registers are unaffected). This routine can be used by CALLing 1D78H or RST 10H. |
| 0018 | (RST 18H) Jumps to 1C90H through 4006H. This routine can be called by using RST 18H or CALL 1C90H. It compares two 16 bit values in HL and DE and sets the S and Z flags accordingly (they are set in the same way as for a normal 8 bit compare). All registers are unchanged except for A. Formula : HL - DE. |
| 0020 | (RST 20H) This routine jumps to 25D9H through 4009H. If the NTF=8 then C=RESET or else C=SET, Z flag will be SET if NTF=3 (S flag is valid also). After execution of RST 20H or CALL 25D9H, A will contain the value NTF-3, all other registers are unchanged. (The NTF is discussed in the arithmetic section). |
| 0028 | (RST 28H) Jumps to 400CH which contains C9H (RET) under Level II BASIC). This vector is only used by Disk BASIC. It is called by |

| LOCATION | DESCRIPTION |
|---|---|
| | the BREAK key routine, and can be used to intercept the BREAK key logic. |
| 002B | Keyboard scanning routine. After CALLing 2BH, the A register will contain the ASCII value for the key that was pressed. The A register will contain 0 if no key was pressed at the time. Apart from the AF register pair the DE register pair is also used by the routine. |
| 0030 | (RST 30H) This location passes control to 400FH which contains a RET (C9H) under Level II. This location is only used by a Disk system. (it is used by the DEBUG utility). |
| 0033 | Character print routine. A CALL 33H will print a character at the current cursor position. The A register must contain the ASCII code for the character or graphics figure that is to be printed before CALLing this routine. The DE register is used by the routine. |
| 0038 | (RST 38H) This location passes control to 4012H. It is used only by a Disk system for the real time clock interrupt service routine. |
| 003B | Character LPRINT routine. Same as 33H but outputs to line printer. (Contents of A register will be printed). |
| 0040 | This location jumps to 5D9H therefore the same rules apply here as for the 5D9H line input routine. |

| LOCATION | DESCRIPTION |
|---|---|
| 0049 | Character input routine. This routine is the same as 2BH except that it will not return until a key is pressed, the character is returned in the A register (AF and DE used). This routine can be used to pause until a key is pressed. |
| 0050 | RS-232 character input routine. (DE is used) The character will be returned in the A register, provided that the RS-232 port has been initialized. |
| 0055 | RS-232 character output routine. (DE is used) The character in the A register will be sent, provided that the RS-232 port has been initialized. |
| 005A | (Model 3 only) RS-232 initialization routine. Load 41F8H, 41FAH and 41F9H with send/receive code, wait/no wait code and characteristics. (see Model 3 manual) Before calling this routine. |
| 0060 | This is a delay routine. The BC register pair is used as the loop counter. The duration of the delay, in microseconds, is the value of BC times 14.65. Register A is used. |
| 0066 | This is the location to which program control jumps when the RESET button is pressed (Non Maskable Interrupt address). |

| LOCATION | DESCRIPTION |
|---|---|
| 0069-0074 | (Model 1 only) This part of the initialization checks to see if a disk drive is connected. If so, it will jump to 0000H. (This is why the reset button will reinitialize DOS but will not return to MEMORY SIZE). |
| 0069 | (Model 3 only) This routine resets all I/O drivers to their normal states. |
| 006C | (Model 3 only) $ROUTE routine. This routine alows the user to "route" data from one device to another. (i.e. keyboard to printer instead of the screen). Load 4222H with two character abbreviation (KI=keyboard, DO=display, PR=printer, RI=RS-232 input, RO=RS-232 output.) for source and 4220H with destination device.(2 characters using ASCII code will take two bytes. To find ASCII code for characters see table in appendix) |
| 0075 | (Model 1 only) This is part of the Level II initialization procedure. Control is passed to this section by the routine at 0696H if there is no disk controller present. It moves a block of memory from 18F7H to 191EH up to 4080H to 40A7H. In other words it sets up the reserved RAM area. |
| 008B | This loads 40A7H with the I/O buffer location address 41E8H. (40A7H is the I/O buffer pointer and can be changed to relocate the buffer). |

| LOCATION | DESCRIPTION |
|---|---|
| 0091-0104 | The rest of the initialization routine. Asks MEMORY SIZE ?, sets the memory pointers accordingly and prints RADIO SHACK LEVEL II BASIC, it then jumps to 1A19H which is the entry point for the BASIC command mode. |
| 00EF | If the HL register pair is loaded with the desired memory size a JUMP to this location will set all required pointers for memory size, stack and variables. |
| 0105 | The "MEMORY SIZE" message is located here. |
| 0111 | The "RADIO SHACK LEVEL II BASIC" message is located here. |
| 012D | This is the entry point for L3 ERROR. |
| 0132 | The POINT routine starts here. |
| 0135 | The SET routine is located here. |
| 0138 | The RESET routine is here. See Part 1 for more details on graphics. |
| 0150 | This is a suitable entry point for the graphics routines. (See Part 1). |
| 01C9 | A CALL 1C9H will clear the screen. (CLS) |
| 01D3 | This is part of the RANDOM routine which takes a value out of the REFRESH register, stores it in location 40ABH and then returns. |

| LOCATION | DESCRIPTION |
|---|---|
| 01D9 | (Model 3 only) Dumps screen contents to printer. |
| 01DC | (Model 3 only) Dumps screen contents from the screen location pointed to by HL to the printer. |
| 01F8 | This routine will turn the cassette recorder off. |
| 0212 | (Model 1 only, not necessary for Model 3) CALL 212H will define which cassette is to be used. Put 00H in the A register to turn on cassette 1, or 0FFH to turn on cassette 2. (BC, DE and HL are unchanged). |
| 021B | (Model 3 only) This routine displays a string of ASCII data to the screen. The string must be terminated with a byte 03 to stop the cursor from scrolling to the start of the next line after printing the string or with 0DH if the cursor must scroll down. |
| 022C | Blinks asterisk in top right corner. This can be used as a subroutine. AF register pair is used. |
| 0235 | This routine reads a byte from tape. A CALL 235H will return with the byte read from tape in the A register BC, DE and HL are unchanged. |
| 0264 | Writes the byte in the A register to tape. BC, DE and HL are unchanged by a CALL 264H. |

| LOCATION | DESCRIPTION |
|---|---|
| 0287 | Writes tape leader and the 0A5H sync byte. DE and HL are unchanged. |
| 028D | (Model 3 only) Scan keyboard for break key, on return if NZ break key is down. If Z break key is NOT down. |
| 0296 | Reads from tape until the leader is found, then keeps going until it is bypassed and the sync byte (A5H) is found, when it returns. DE, BC and HL are unchanged by this. |
| 0298 | (Model 3 only) A CALL 298H will turn the clock on. A is used. |
| 029F | (Model 1 only) Places the double asterisks in the right top corner to show that the sync byte has been found. |
| 02A1 | (Model 3 only) A CALL 2A1H will turn the clock off. A is used. |
| 02B5 | This location passes control to the routine used by the BASIC command SYSTEM. |
| 0314 | This routine reads two bytes from tape (providing that the tape is already running) and puts them in the HL register pair. It is used by the SYSTEM routine to read the last two bytes on tape which give the entry point. A JP (HL) can then be executed to jump to the location specified, when used for this purpose. Only HL is used by this routine. |

| LOCATION | DESCRIPTION |
|---|---|
| 032A | This is a general purpose output routine which outputs a byte from the A register to video, tape or printer. In order to use it, the location 409CH must be loaded with -1 for tape, 0 for video or 1 for the line printer. |
| 033A | A Print routine which performs the same function as 33H except that it does not destroy the content of the DE register pair. This means that all the general purpose registers are saved, which is desirable. |
| 035B | Here is the routine to simulate the INKEY$ function. It performs exactly the same function as 2BH but it restores all registers, whereas 2BH destroys the contents of the DE register pair. This makes 35BH more useful than 2BH. |
| 0361 | This is one of the general purpose input routines (see 5D9 and 1BB3 also). This routine inputs a string from the keyboard up to a maximum of 240 characters (0F0H), and echoes them to the screen. It puts this data into a buffer located at the address pointed to by the buffer pointer at 40A7H (e.g. if 40A7H contains 5000H the data will be stored from 5000H onwards). The string is terminated with a zero byte. The program returns from this routine as soon as the ENTER key has been pressed. When it does so, HL contains the start address of the input string. (RST 10H can be used to make HL point to the first character of the string, if required). |

| LOCATION | DESCRIPTION |
|---|---|
| 039C | This is the LPRINT routine. All registers are saved. The byte to be printed should be in the A register. |
| 03E3 | (Model 1 only - Model 3 use 3024H) This is the keyboard driver. It scans the keyboard and converts the bit pattern obtained to ASCII and stores it in the A register. |
| 0458 | (Model 1 only - Model 3 use 473H) This is the video driver. The character to be displayed should be in the C register. This routine handles scrolling etc. |
| 04C3 | (Model 1 only - Model 3 use byte at 4210H) Changes display to 64 character mode (A register is used.) |
| 04F6 | (Model 1 only - Model 3 use byte at 4210H) Changes display to 32 character mode. A and HL registers used. |
| 057C | (Model 1 only - Model 3 use 5C5H) Clear to end of frame routine. To use this routine load the HL register pair with the screen address from which you want the erasing to start. The DE and A registers are used. |
| 058D | (Model 1 only - Model 3 use 3C2H) LPRINT driver routine, handling printer I/O etc. The character to be printed should be in register C. |
| 05D9 | This is the most basic of the string input routines and is used by the two others |

| LOCATION | DESCRIPTION |
|---|---|
| | (1BB3H and 0361H) as a subroutine. To use it, load HL with the required buffer address and the B register with the maximum buffer length required. Keyboard input over the specified maximum buffer length is ignored, and after pressing the (ENTER) key it will return with HL containing the original buffer address and B with the string length. |
| 0674-069E | (Model 1 only) This is the start of the main initialization routine. The interpreter arrives at this location from 0002H, at 0696H a test is made to see if the floppy disk controller is present if it is not the interpreter will jump to 0075H to initialize LEVEL II BASIC. |
| 069F-06CB | (Model 1 only)This routine will load the BOOT/SYS program located on the DOS diskette. The program is 255 bytes long and is loaded to 4200H, as soon as this is done the BOOT program will be allowed to continue initialization to the DOS READY mode. |
| 06CC | (Model 1 only) This is an alternative re-entry point into BASIC. A jump to 6CCH is better than a jump to 1A19H as the latter can cause out of memory errors (OM ERROR) etc. |
| 070B | Single-precision addition (ACC=(HL)+ACC) involving a buffer pointed to by the HL register pair and the ACC (see arithmetic section in Part 1 of this manual for information on the ACC). This part of the program loads the BCDE registers with the |

| LOCATION | DESCRIPTION |
|---|---|
| | value from the buffer, then passes control to 716H. |
| 0710 | Single-precision subtraction (ACC=(HL)-ACC). This loads the BCDE registers with the value from (HL), then passes control to 713H. |
| 0713 | Single-precision subtraction (ACC=BCDE-ACC). The routine actually inverts the ACC and adds it to the contents of the BCDE registers which, in effect, is a subtraction. The result will be stored in the arithmetic work area (ACC). |
| 0716 | Single-precision addition (ACC=BCDE+ACC). This routine adds two single-precision values and stores the result in the ACC area. |
| 07B2 | This is the OV ERROR entry point. |
| 0809 | LOG routine, (ACC=LOG (ACC)). This routine finds the LOGarithm of the value in the ACC area. |
| 0847 | Single-precision multiplication (ACC=BCDE*ACC). |
| 08A2 | Single-precision division (ACC=BCDE/ACC). If ACC=0 a " /0 ERROR " will result. |
| 0955 | Checks if ACC=0. If so, the Z flag will be set. |

| LOCATION | DESCRIPTION |
|---|---|
| 0977 | ABS routine (ACC=ABS(ACC)) input and output can be integer, single-precision or double-precision, depending on what is placed in the NTF (NTF=2, 4 or 8). (For a definition of NTF, see Part 1). |
| 0982 | NEGATE function for single-precision values (ACC=-ACC). Only BC and DE are saved. |
| 098A | SGN function (ACC=SGN(ACC)). After execution, NTF=2 and ACC=-1, 0 or 1 depending on sign and value of ACC before execution. |
| 0994 | This routine checks the sign of the ACC. NTF must be set. After execution A register=00 if ACC=0, A=01 if ACC is greater than 0 or A=FFH if A is less than 1. The Flags are also valid. |
| 09A4 | Loads single-precision value from ACC to stack ((SP)=ACC). To retrieve this value, POP BC followed by POP DE. A, BC and HL are unchanged by this function. |
| 09B1 | This routine loads four bytes from the location pointed to by HL, into the ACC. (ACC=(HL)). |
| 09B4 | This routine loads the ACC with the contents of the BC and DE register pairs. (ACC=BCDE). BC and HL remain unaltered. |
| 09BF | This routine is the opposite of the 9B4H routine. It loads four bytes from the ACC (single-precision) into the BC and DE |

| LOCATION | DESCRIPTION |
|---|---|
| | register pairs. (BCDE=ACC). A is unchanged. |
| 09C2 | This routine loads the BCDE register pairs with four bytes from the location pointed to by HL. (BCDE=(HL)). With these types of data movements, the E register is loaded with the LSB and the B register with the MSB. |
| 09CB | This routine is the opposite of the 9B1H routine. It loads four bytes from the ACC to the memory location pointed to by HL. ((HL)=ACC). |
| 09CE | Data move routine. This moves four bytes from the location pointed to by DE into the location pointed to by HL. ((HL)=(DE)). |
| 09D2 | Data move routine. The location pointed to by DE is loaded with bytes from the location pointed to by HL. The number of bytes moved is determined by the value in the NTF. ((DE)=(HL)). |
| 09D3 | This routine is similar to 9D2H above. The only difference is that it moves data in the opposite direction ((HL)=(DE)). |
| 09D6 | This routine is the same as 9D3H except that the number of bytes moved depends on the value in the A register ((HL)=(DE)). |
| 09D7 | This routine is the same as 9D6H except that the number of bytes shifted is determined by the value in the B register ((HL)=(DE)). |

| LOCATION | DESCRIPTION |
|---|---|
| 09F4 | This routine is used by the double-precision logic. It moves a number of bytes (the number depending on the value stored in the NTF) from the AACC into the ACC. ((ACC)=(AACC)). |
| 09FC | This is the opposite of 9F4H. ((AACC)=(ACC)). |
| 0A0C | Single-precision compare. Compares the ACC with the contents of BCDE registers. After execution of this routine, the A register will contain: A=0 if ACC=BCDE, A=1 if ACC is greater than BCDE or A=FFH if ACC is less than BCDE. |
| 0A39 | Integer compare. Compares HL with DE. After execution, A=0 if HL=DE, A=1 if HL is greater than DE or A=FFH if HL is less than DE. The S and Z flags are valid. |
| 0A4F | Double-precision compare. Compares the ACC with the AACC. After execution the A register will contain: A=0 if ACC=AACC, A=1 if ACC is greater than AACC or A=FFH if ACC is less than AACC. S and Z flags are valid. |
| 0A78 | Double-precision compare. This compare is the opposite of the A4FH compare. It compares the AACC with the ACC. (Remember that a compare is actually a subtraction which is never executed, therefore a compare can be done in two ways with the same values (A-B and B-A) ). The results are the same as the A4FH routine. |

| LOCATION | DESCRIPTION |
|---|---|
| 0A7F | CINT routine. Takes a value from ACC, converts it to an integer value and puts it back into the ACC. On completion, the HL register pair contains the LSB of the integer value, and the NTF contains 2 (Integer=2). If NTF=3 (string) a TM ERROR will be generated and control will be passed to BASIC. |
| 0A8A | Same as 0A7F. |
| 0A9A | This is the routine that returns the value in the HL register pair to the BASIC program that called it. In effect, it moves the content of HL into the ACC (ACC=HL). |
| 0A9D | Set NTF to Integer (2). (A=used). |
| 0AB1 | CSNG routine. Takes value from ACC and converts it to single-precision. The result is put in ACC and NTF contains 4. |
| 0ADB | CDBL routine. Takes a value from ACC and converts it to double-precision. The result will be in ACC and NTF will be 8. |
| 0AF4 | This routine calls 20H (RST 20H) and returns if NTF=3 (string) else if NTF is not 3 then it generates a TM ERROR. BC, DE, and HL are saved. |
| 0AF6 | This is the entry point for the TM ERROR. |

| LOCATION | DESCRIPTION |
|---|---|
| 0AFB | This routine will reset the BC and DE register pairs if the A register contains 0. (XOR A before calling this routine). |
| 0B26 | FIX routine. Takes a value from ACC and converts it to an integer value. The result will be in ACC. NTF will be 2 if value is smaller than 32767 else it will be 4. An error will be generated if NTF=3 (string). |
| 0B37 | Same as FIX (B26H). |
| 0BD2 | Integer addition (ACC=DE+HL). After execution NTF=2, or 4 if overflow has occurred, in which case the result in the ACC will be single-precision. |
| 0BC7 | Integer subtract (ACC=DE-HL). The result is returned in both the ACC and the HL register pair. |
| 0BF2 | Integer multiply. (ACC=DE*HL) (rules same as above). |
| 0C51 | Negate HL routine. This routine changes the sign of the HL register pair and stores it in the ACC. (HL=ACC=-HL). The result is returned in both the HL register pair and the ACC. |
| 0C70 | Double-precision subtraction (ACC=ACC-AACC). |
| 0C77 | Double-precision addition (ACC=ACC+AACC). |
| 0DA1 | Double-precision multiplication (ACC=ACC*AACC). |

| LOCATION | DESCRIPTION |
|---|---|
| 0DE5 | Double-precision division (ACC=ACC / AACC). |
| 0E65 | This routine converts an ASCII string (pointed to by HL) to a double-precision value and stores it in the ACC. The NTF is fixed accordingly. The string must be terminated with a comma or zero byte. Note that the AACC is destroyed in the process and that HL will point to the delimiter at the end of the string. The string formats must follow the same rules as in BASIC. |
| 0E6C | This routine is the same as 0E65H above, except that it fixes the ACC and the NTF to the smallest possible number type. |
| 0FAF | Converts the contents of HL to an ASCII coded numeric string. The string will be in the ACC on return. |
| 0FBD | Conversion routine. Converts the value from the ACC to an ASCII string delimited with a zero byte. The number type can be any of Integer, Single or Double-precision. After execution HL will be pointing to the start of the string. The ACC and AACC are destroyed by the process. |
| 13E7 | SQR routine. Single precision only. (ACC = SQR (ACC)). |
| 1439 | EXP routine. Single precision only. (ACC = EXP (ACC)). |

| LOCATION | DESCRIPTION |
|---|---|
| 14C9 | RND routine. Integer, Single or Double precision. Output will be single precision. (ACC = RND (ACC)). |
| 1541 | COS routine. Single precision only. (ACC = COS (ACC)). |
| 1547 | SIN routine. Single precision only. (ACC = SIN (ACC)). |
| 15A8 | TAN routine. Single precision only. (ACC = TAN (ACC)). |
| 15BD | ATN routine. Single precision only. (ACC = ATN (ACC)). |
| 197A | OM ERROR entry point. |
| 1997 | SN ERROR entry point. |
| 199A | /0 ERROR entry point. |
| 199D | NF ERROR entry point. |
| 19A0 | RW ERROR entry point. |
| 1A19 | Re-entry point into BASIC command mode entry point. (see 6CCH also). |
| 1AF8 | This routine fixes the line pointers in a BASIC program. This is useful if, for instance, you need to move BASIC program lines in memory. A CALL 1AF8 will then restore the line pointers. Registers A, HL and DE are used. |

| LOCATION | DESCRIPTION |
|---|---|
| 1B2C | This routine searches a BASIC program for a BASIC line with a line number matching the value in the DE register pair. Therefore, to use this routine, the required line number must be placed in the DE register pair. When a match is found, this routine sets the carry flag, the BC register pair will be loaded with the start of the required BASIC line, the HL register pair points to the start of the next line. HL, AF and BC are used. |
| 1B49 | Entry point of the NEW command. |
| 1B4D | A NEW without clearing the screen can be done from a machine language routine by calling 1B4DH. This routine can also be used to reset the BASIC program pointers to any location in BASIC RAM. To do this load the start of BASIC pointer (40A4H) with the new start address and CALL 1B4DH to reset all the pointers. |
| 1B5D | From here you can RUN a BASIC program to do this use the following procedure:<br><br>`LD HL,1D1EH`<br>`PUSH HL`<br>`JP 1B5DH` |
| 1B6E | A CALL to this location will reset most BASIC pointers in reserved RAM. |
| 1BB3 | This is the last of the general purpose input routines. This routine functions identically to the routine at 0361H, with |

| LOCATION | DESCRIPTION |
|---|---|
| | the exception that it prints a "?" on the screen like INPUT does with BASIC, before allowing input from the keyboard. |
| 1BC0 | This routine compresses BASIC command strings to their tokens i.e. END would be compressed to 80H. |
| 1C90 | The RST 18H code is located here. |
| 1C96 | The RST 8H code is located here. |
| 1CA1 | FOR entry point. |
| 1D5A | The actual BASIC interpreter is located here. HL should be pointing to the BASIC text to be interpreted. |
| 1D78 | The RST 10H code is located here. |
| 1D91 | RESTORE logic is located here. |
| 1DA9 | STOP entry point. |
| 1DAE | END entry point. |
| 1DE4 | CONT entry point. |
| 1DF7 | TRON entry point. |
| 1DF8 | TROFF entry point. |
| 1E00 | DEFSTR entry point. |
| 1E03 | DEFINT entry point. |

| LOCATION | DESCRIPTION |
|---|---|
| 1E06 | DEFSNG entry point. |
| 1E09 | DEFDBL entry point. |
| 1E3D | This routine tests the value pointed to by the HL register pair and sets the carry flag if it is an ASCII letter value. Otherwise it resets the carry flag. |
| 1E4A | FD ERROR entry point. |
| 1E5A | Converts numeric ASCII string pointed to by the HL register pair, to Hexadecimal and places the result in the DE register pair. After execution HL points to the delimiter and the A register contains the delimiter value. The Z flag is set if the delimiter = 00H or 3AH. Z is reset if any other delimiter is used. If there is no string at the location pointed to by the HL register pair the routine will return a MO ERROR (missing operand error). If the result exceeds 0FFFFH an OV ERROR (overflow) results. |
| 1E7A | Location of CLEAR logic. |
| 1EA3 | RUN initialization logic. |
| 1EB1 | GOSUB entry point. |
| 1EC2 | GOTO entry point. |
| 1EDE | RETURN entry point. |
| 1EEC | RG ERROR entry point. |

| LOCATION | DESCRIPTION |
|---|---|
| 1F05 | DATA entry point. |
| 1FF4 | ERROR entry point. |
| 2003 | UE ERROR entry point. |
| 2039 | IF entry point. |
| 2067 | LPRINT logic. |
| 206F | PRINT logic. |
| 2076 | PRINT @ logic. |
| 2137 | TAB logic. |
| 2170 | This routine will set the DEVICE TYPE FLAG (409CH) to zero (video output). |
| 2178 | ?REDO message string. |
| 219A | INPUT logic. |
| 21C9 | INPUT routine can be CALLed from here. To use load HL with string address and A with the first character. For example to execute INPUT "AMOUNT";X use the following procedure: |

```
INPUT     DEFM     ""AMOUNT";X"
```

| LOCATION | DESCRIPTION |
|---|---|
| | DEFB 0 |
| | LD HL,INPUT |
| | LD A,(HL) |
| | CALL 21C9H |
| | The only other requirement is that the variable (X in this case) must already exist. |
| 21EF | READ logic. |
| 2286 | ?EXTRA IGNORED. message string. |
| 22B6 | NEXT logic. |
| 2337 | This routine evaluates a BASIC expression pointed to by the HL register pair and stores the result in the ACC. The expression must be terminated with a zero byte, comma, right bracket or colon. After execution, HL will point to the delimiter and in the case of string expressions, the ACC Will contain the address of the first of three bytes that contain string length and string address. Note that the stack is used frequently and the machine should be in RUN mode in order to use this routine. (See sample program in Appendix 1 for an application of this routine). |

| LOCATION | DESCRIPTION |
|---|---|
| 2490 | Integer divide. (ACC = DE / HL) The result will be in Single precision (NTF = 4) and will be in the ACC. |
| 24CF | ERR logic. |
| 24DD | ERL logic. |
| 24EB | VARPTR logic. |
| 2540 | This routine loads a variable to the ACC and sets the NTF. The HL register pair must point to the ASCII variable name. After execution the HL register pair will point to the character following the last character of the variable used. The value of the variable will be loaded in the ACC. For strings however (NTF = 3), The ACC will contain the address of the three bytes which contain the string length and the actual string address (see LEVEL II BASIC MANUAL). Also note that if the variable cannot be found it will be created and given a value of zero. |
| 25D9 | The RST 20 code is located here. |
| 25F7 | OR logic. |
| 25FD | AND logic. |
| 2608 | DIM logic. |

| LOCATION | DESCRIPTION |
|---|---|
| 260D | This is the variable location and creation logic. This routine returns the address of a variable in memory or creates it if it is not found. In order to use this routine, the HL register pair must point to the variable name (ASCII). Then, after execution, HL will point to the character following the variable name and the location of the variable will be returned in the DE register pair. For integer, single or double-precision (NTF=2, 4 or 8); the address returned in DE will be the same as for the VARPTR command under BASIC. (See Level II BASIC manual on VARPTR). For strings (NTF=3) however, the address returned in DE will point to the first of three bytes containing the string length and string address. |
| 273D | BS ERROR entry point. |
| 27C9 | MEM logic. |
| 27D4 | FRE logic |
| 27F5 | POS logic. |
| 27FE | USR ;logic. |
| 2831 | ID ERROR entry point. |
| 2836 | STR$ logic. |
| 28A1 | ST ERROR entry point. |

| LOCATION | DESCRIPTION |
|---|---|
| 28A7 | This is a general purpose output routine. It will output data to the display, printer or cassette, depending on the contents of 409CH. (0=video, -1=tape, 1=printer). The address of the first character in the string to be output must be in the HL register pair, and the string must end with a zero byte or a quote (22H). |
| 29D7 | This routine sets the HL register pair to point to the data in the ACC. |
| 2A03 | LEN logic. |
| 2A0F | ASC logic. |
| 2A1F | CHR$ logic. |
| 2A2F | STRING$ logic. |
| 2A61 | LEFT$ logic. |
| 2A91 | RIGHT$ logic. |
| 2A9A | MID$ logic. |
| 2AC5 | VAL logic. |
| 2AEF | INP logic. |
| 2AFB | OUT logic. |
| 2B01 | STEP logic. |
| 2B05 | This routine takes the value from the ACC, converts it to an integer value and places |

| LOCATION | DESCRIPTION |
|---|---|
| | the result in the DE register pair. The Z flag will be set if the result in DE is smaller than or equal to 255 (FFH). (DE = INT (ACC)). |
| 2B1C | This routine converts a numeric ASCII string pointed to by the HL register pair into a hexadecimal value and places the result in the A register. If the result is larger than 255 (FFH) then an FC ERROR (illegal function call) will be generated. After execution the HL register pair will point to the delimiter. If the delimiter is a zero byte or a colon (0AH) then the Z flag will be set. Any other delimiter will cause the Z flag to be reset. |
| 2B29 | LLIST logic. |
| 2B2E | LIST logic. |
| 2B7E | LIST routine. |
| 2BC6 | DELETE logic. |
| 2BF5 | CSAVE routine. Load HL with the address of the program name. The program name is an ASCII string delimited with a double quote. (only the first letter of the program name is used). |
| 2C1F | CLOAD logic. |
| 2CAA | PEEK logic. |
| 2CA5 | "BAD" message string. |

| LOCATION | DESCRIPTION |
|---|---|
| 2CB1 | POKE logic. |
| 2CBD | USING logic. |
| 2E60 | EDIT logic. |
| 2F0A | This routine will output a string of data to the display, printer or tape, again depending on 409CH. The difference between 28A7H above and this routine is that the D and B registers play a role. The D register should be loaded with the maximum number of characters that should be printed. The routine will stop when either the maximum number of characters has been printed or if the delimiter is reached which should be a ZERO byte. If the B register is set to zero before this routine is called the B register will reflect the actual ammount of characters printed. Otherwise it will simply be incremented by this amount. |

LOCATION DESCRIPTION

* RESERVED RAM AND DEVICE ADDRESSES FOR THE MODEL 1 *

| LOCATION | USE OR COMMENTS |
|---|---|
| 3000-37DD | This area is set aside for future DMA devices, there is nothing here at all. This area can be used for custom interfaces. |
| 37DE | DOS status address. These DOS locations are not memory but communication addresses which communicate directly or indirectly with the floppy disk controller IC. (For more information on the floppy disk controller see the FD 1771 floppy disk controller data sheet from WESTERN DIGITAL). |
| 37DF | DOS communication data address. |
| 37E0 | Interrupt latch address. |
| 37E1 | Disk drive select latch address for drive 0. |
| 37E2 | Cassette drive latch address. |
| 37E3 | Disk drive latch address for drive 1. |
| 37E4 | Cassette select address defined by 212H. |
| 37E5 | Disk drive latch address for drive 2. |
| 37E7 | Disk drive latch address for drive 3. |
| 37E8 | Line printer port address. |
| 37EC-37EF | Floppy disk controller addresses. |

| LOCATION | DESCRIPTION |
|---|---|
| | --** MODEL 1 & 3 RESERVED RAM ADDRESSES **-- |
| | (These addresses are in common between the Model 1 and the Model 3). |
| 3801-3880 | Keyboard area. (See special section on keyboard for more information). |
| 3C00-3FFF | Video display memory. |
| 4000 | Jump vector for RST 8H. |
| 4003 | Jump vector for RST 10H. |
| 4006 | Jump vector for RST 18H. |
| 4009 | Jump vector for RST 20H. |
| 400C | Jump vector for RST 28H. |
| 400F | Jump vector for RST 30H. |
| 4012 | Jump vector for RST 38H. |
| | --- KEYBOARD DATA CONTROL BLOCK --- |
| 4015 | Device type. (The following data for Model 1 only) If this location is loaded with zero a jump to 4033H will occur every time the keyboard is scanned. |
| 4016 | Driver address. The contents of this address and the next one contains the address to which the keyboard scanning |

| LOCATION | DESCRIPTION |
|---|---|
| | routine will jump each time it scans the keyboard. |
| | --- VIDEO DISPLAY DATA CONTROL BLOCK --- |
| 401D | Device type. |
| 401E | Driver address. |
| 4020 | Location of cursor in video memory. (Two byte address). |
| 4023 | Cursor character. |
| | --- LINE PRINTER CONTROL BLOCK --- |
| 4025 | Device type. |
| 4026 | Driver address. If this driver address is changed to the driver address of the video control block all LPRINT commands will print to the display instead of the line printer, and vice versa. |
| 4028 | Number of lines per page plus 1. |
| 4029 | Number of lines printed plus 1. |
| 402B | Line printer max. line length less 2. |
| 4033 | (Model 1 only) A jump to a machine language routine can be placed here. (See section on program intercept and 4015H). |

| LOCATION | DESCRIPTION |
|---|---|
| 4036-403C | Small buffer for keyboard decoding routine (used for keyboard rollover). |
| 403D | (Model 1 only) Print size flag. (0=64 characters, 8=32 characters). |
| 4041-4046 | TIME$ storage area for 25 ms counts, seconds, minutes, hours, year, day and month respectively. |
| 408E | Entry pointer for USR routines. |
| 4099 | INKEY$ storage. |
| 409A | Error code for RESUME. |
| 409B | Printer carriage position. |
| 409C | Device type flag (0=video, 1=printer, -1=tape). |
| 409D | Used by PRINT#. |
| 40A0 | String space pointer. |
| 40A2 | Current line being processed by BASIC. |
| 40A4 | Start of BASIC program location. |
| 40A6 | Line cursor position, used by TAB. |
| 40A7 | I/O buffer pointer. |
| 40AA | LSB of seed number for RND. |

| LOCATION | DESCRIPTION |
|---|---|
| 40AB | NSB (Next most Significant Byte) of seed. (See 1D3H). |
| 40AC | MSB of seed. |
| 40AF | NTF this is the Number Type Flag. This address tells BASIC the type of number which is contained in the ACC. (2=integer, 3=string, 4=single and 8=double-precision). |
| 40B1 | Top of BASIC memory pointer. (MEM SIZE) |
| 40B3 | String work area pointer. |
| 40B5 | Usual string work area. |
| 40D6 | String space pointer (current location). |
| 40DC | Used by DIM. |
| 40DE | Used by PRINT USING. |
| 40DF | Entry point storage for SYSTEM programs. |
| 40E1 | AUTO flag (0=auto off, else auto on). |
| 40E2 | Current line number (used by AUTO). |
| 40E4 | Increment size for AUTO. |
| 40E6 | Points to the location in memory of the BASIC program material which the interpreter is currently processing. |
| 40E8 | Stack pointer. |

| LOCATION | DESCRIPTION |
|---|---|
| 40EA | Used by RESUME to store error line number. |
| 40EC | EDIT line number. |
| 40EE | Used by RESUME. Stores error line number. |
| 40F0 | Contains BASIC line number of line containing error handler. |
| 40F2 | Error flag. |
| 40F5 | Last line executed. |
| 40F7 | Used by CONTinue. |
| 40F9 | Simple variables pointer and end of BASIC program pointer. |
| 40FB | Array pointer. |
| 40FD | Free space pointer. |
| 40FF | Data pointer. Points to current data item. |
| 4101-411A | Variable type declaration table. 2=INT, 3=String, 4=single, 8=Double. |
| 411B | TRON flag 0=TROFF. |
| 411D-4124 | ACC (ACCumulator area). See arithmetic section for more information. |
| 4127-412E | AACC (Auxiliary ACCumulator area). |
| 4130 | Line number work area pointer. |

| LOCATION | DESCRIPTION |
|---|---|
| --- DOS COMMAND ENTRY POINT TABLE --- | |
| 4152 | CVI |
| 4155 | FN |
| 4158 | CVS |
| 415B | DEF |
| 415E | CVD |
| 4161 | EOF |
| 4164 | LOC |
| 4167 | LOF |
| 416A | MKS$ |
| 416D | MKS$ |
| 4170 | MKD$ |
| 4173 | CMD |
| 4176 | TIME$ |
| 4179 | OPEN |
| 417C | FIELD |
| 417F | GET |
| 4182 | PUT |

| LOCATION | DESCRIPTION |
|---|---|
| 4185 | CLOSE |
| 4188 | LOAD |
| 418B | MERGE |
| 418E | NAME |
| 4191 | KILL |
| 4194 | & |
| 4197 | LSET |
| 419A | RSET |
| 419D | INSTR |
| 41A0 | SAVE |
| 41A3 | LINE |
| 41A6 | Error intercept used by Disk BASIC to intercept errors so that they can be printed out in full. |
| 41A9 | Used by Disk BASIC to support its additional USR functions. |
| 41AC | This address is called just before READY is displayed on screen. |
| 41AF | Intercept for the general purpose input routine (0361H). |

| LOCATION | DESCRIPTION |
|---|---|
| 41B2 | This intercept address can be used to check for encoded (compressed) BASIC commands. It is called before the command is executed. |
| 41B5 | This location is called when the resident BASIC program needs to be altered. For instance if a new line is typed in, before it is inserted in the BASIC program this location is called. |
| 41BB | Intercept for program initialisation. |
| 41BE | This address is called after the Device type flag at 409CH is set to 0 (i.e. video output) by 2170H. |
| 41C1 | Intercept for general purpose output routine (032AH). |
| 41C4 | Called when BASIC scans keyboard. INKEY$ etc. |
| 41C7 | Intercept for RUN. |
| 41CA | Called by PRINT routine, before anything is printed. |
| 41D6 | Called before INPUT. |
| 41D9 | Used by disk BASIC to provide extra MID$ functions. |
| 41DF | Intercept for LIST, HL points to LINENUMBER, BC to next LINEPOINTER, DE will contain maximum line number to be listed if it has been specified in the LIST command. |

| LOCATION | DESCRIPTION |
|---|---|
| 41E2 | This location is called just before the *? prompt is printed by the SYSTEM routine. This can be used to intercept the SYSTEM routine after it has loaded a tape, by putting a jump back into your routine at this location and executing a CALL 2B5H. |
| 41E8-42E8 | I/O Buffer area. |
| 42E9 | (43E9 for Model 3) Level II BASIC programs start here. |
| 6A24 | (Model 1 only) Disk BASIC programs start here. |

| LOCATION | DESCRIPTION |
|---|---|

--** MODEL 3 EXTRA ROM ADDRESSES **--

The Model 3 has nearly 2K bytes extra ROM added to the interpreter, this section starting at 3000H mainly contains I/O routines for cassette, keyboard and RS232. Note that this section will reveal where the cassette routines are, it is however not recommended that these routines are used instead of the normal cassette routines, both for compatibillity with the Model 1 and in case the ROMS are altered in future which means that these routines might be relocated.

| | |
|---|---|
| 3000 | Write leader and sync. byte (500 baud) |
| 3003 | Write leader and sync. byte (1500 baud) |
| 3006 | Read leader till sync byte found (500 baud) |
| 3009 | Read leader till sync byte found (1500 baud) |
| 300C | Turn cassette off. |
| 300F | Turn cassette motor on, and wait for motor to reach full speed. |
| 3012 | Check if disk drive ready - then "boot". |
| 3015 | Initialization entry (RESET). |
| 3018 | Interrupt service routine (ISR). |
| 301B | Initialize RS-232. |
| 301E | RS-232 input routine. |
| 3021 | RS-232 output routine. |
| 3024 | Keyboard input routine. |
| 3027 | $ROUTE I/O routine. (see Model 3 manual) |
| 302A | Handles "#" (like in PRINT #) |
| 3033 | $DATE vector. |
| 3036 | $TIME vector. |
| 3042 | Set cass. routine (asks: CASS? ; then after keyboard response sets cassette baud rate. |

| LOCATION | DESCRIPTION |
|---|---|
| | --** MODEL 3 EXTRA RAM ADDRESSES **-- |
| 4019 | Caps lock switch. If 0 allow upper and lowercase, if not 0 uppercase only. |
| 401C | Cursor blink switch. 0 = blink, not 0 is not blink. |
| 403D | Interrupt vector for I/O bus interrupt. |
| 4040 | Interrupt vector for RS-232 error interrupt. |
| 4043 | Interrupt vector for undefined interrupt. |
| 4046 | Interrupt vector for real time clock. |
| | --* MODEL 3 EXTRA DEVICE CONTROL BLOCKS *-- |
| | The following device control blocks have the same basic format and purpose as, for instance, the keyboard control block at 4015H (see 4015H). The first byte is the device type, and the following two bytes form the actual address of the device drivers involved. The addresses for a standard Model 3 are shown in brackets for reference, these might change in future, however. |
| 41E5 | ---- RS-232 INPUT DEVICE CONTROL BLOCK ---- |
| 41E6 | RS-232 input driver address (301EH) |
| 41E8 | One byte buffer for RS-232 input. |
| 41ED | ---- RS-232 OUTPUT DEVICE CONTROL BLOCK ---- |
| 41EE | RS-232 output driver address (3021H) |
| 41F0 | One byte output buffer for RS-232. |

| LOCATION | DESCRIPTION |
|---|---|
| 41F5 | ---- RS-232 INIT. DEVICE CONTROL BLOCK ---- |
| 41F6 | RS-232 init. routine address (301BH) |
| 41F8-41FA | RS-232 config. bytes (see Model 3 manual) |
| 4206 | Interrupt vector for RS-232 transmit register empty interrupt. |
| 421D | ---- ROUTE - DEVICE CONTROL BLOCK ---- |
| 421E | ROUTE - address (3739H) |
| 4209 | Interrupt vector for RS-232 receive register full interrupt. |
| 4210 | Mask byte. (see separate heading for this byte) |
| 4211 | Cassette Baud rate switch 0=500 baud, not 0 =1500 baud. |
| 4212 | Counter for tape routines, when it has counted down to zero will flash an asterisk in the top right corner of the screen. |
| 4213 | Mask for port E0H. |
| 4214 | Video display scroll protect lines from 0 to 7. |
| 4217 | Time and date - 6 bytes, 1 each for : seconds, minutes, hours, year, day, month. |
| 4220 | $ROUTE destination device. (see 6CH) |
| 4222 | $ROUTE source device (see 6CH) |

# APPENDIX 1

## CONVERSION TABLE.

This table lists all possible values of a single byte (0 to 255) and their respective uses in the TRS-80. This includes all control codes (carriage return, line feed etc.) and compression codes for BASIC commands. The BASIC commands are stored in memory as single byte values. The first of these for instance, is the END statement which is stored as 80H. At times we may want to write a machine language program which processes lines in a BASIC program. In order to do this we must know what the compression codes and control codes are; this table will allow the user to find them.

Another application is when use is made of the keyboard scanning routines. In order to make use of these. the programmer needs to know the ASCII codes which each key on the keyboard returns when pressed.

| BINARY | HEX. | DEC. | COMMENTS |
|---|---|---|---|
| 0000 0000 | 00 | 0 | NULL |
| 0000 0001 | 01 | 1 | BREAK KEY |
| 0000 0010 | 02 | 2 | |
| 0000 0011 | 03 | 3 | |
| 0000 0100 | 04 | 4 | |
| 0000 0101 | 05 | 5 | |
| 0000 0110 | 06 | 6 | |
| 0000 0111 | 07 | 7 | |
| 0000 1000 | 08 | 8 | BACK SPACE (left arrow key) |
| 0000 1001 | 09 | 9 | TAB (right arrow key) |
| 0000 1010 | 0A | 10 | LINE FEED (down arrow, CTRL) |
| 0000 1011 | 0B | 11 | |
| 0000 1100 | 0C | 12 | FORM FEED |
| 0000 1101 | 0D | 13 | CARRIAGE RET.(enter, newline) |
| 0000 1110 | 0E | 14 | |
| 0000 1111 | 0F | 15 | |
| 0001 0000 | 10 | 16 | |
| 0001 0001 | 11 | 17 | |
| 0001 0010 | 12 | 18 | |
| 0001 0011 | 13 | 19 | |
| 0001 0100 | 14 | 20 | |
| 0001 0101 | 15 | 21 | |
| 0001 0110 | 16 | 22 | |
| 0001 0111 | 17 | 23 | CHANGE TO 32 CHAR. MODE |
| 0001 1000 | 18 | 24 | ERASE LINE (shift+left arrow) |
| 0001 1001 | 19 | 25 | (shifted right arrow key) |
| 0001 1010 | 1A | 26 | (shifted down arrow key) |
| 0001 1011 | 1B | 27 | (shifted up arrow key) |
| 0001 1100 | 1C | 28 | HOME CURSOR |
| 0001 1101 | 1D | 29 | |
| 0001 1110 | 1E | 30 | |
| 0001 1111 | 1F | 31 | CLEAR |
| 0010 0000 | 20 | 32 | SPACE |
| 0010 0001 | 21 | 33 | ! |
| 0010 0010 | 22 | 34 | " |

| BINARY | HEX. | DEC. | COMMENTS |
|---|---|---|---|
| 0010 0011 | 23 | 35 | # |
| 0010 0100 | 24 | 36 | $ |
| 0010 0101 | 25 | 37 | % |
| 0010 0110 | 26 | 38 | & |
| 0010 0111 | 27 | 39 | SINGLE QUOTE. |
| 0010 1000 | 28 | 40 | ( |
| 0010 1001 | 29 | 41 | ) |
| 0010 1010 | 2A | 42 | * |
| 0010 1011 | 2B | 43 | + |
| 0010 1100 | 2C | 44 | , |
| 0010 1101 | 2D | 45 | - |
| 0010 1110 | 2E | 46 | . |
| 0010 1111 | 2F | 47 | / |
| 0011 0000 | 30 | 48 | 0 |
| 0011 0001 | 31 | 49 | 1 |
| 0011 0010 | 32 | 50 | 2 |
| 0011 0011 | 33 | 51 | 3 |
| 0011 0100 | 34 | 52 | 4 |
| 0011 0101 | 35 | 53 | 5 |
| 0011 0110 | 36 | 54 | 6 |
| 0011 0111 | 37 | 55 | 7 |
| 0011 1000 | 38 | 56 | 8 |
| 0011 1001 | 39 | 57 | 9 |
| 0011 1010 | 3A | 58 | : |
| 0011 1011 | 3B | 59 | ; |
| 0011 1100 | 3C | 60 | LESS THAN SYMBOL |
| 0011 1101 | 3D | 61 | = |
| 0011 1110 | 3E | 62 | GREATER THAN SYMBOL |
| 0011 1111 | 3F | 63 | ? |
| 0100 0000 | 40 | 64 | @ |
| 0100 0001 | 41 | 65 | A |
| 0100 0010 | 42 | 66 | B |
| 0100 0011 | 43 | 67 | C |
| 0100 0100 | 44 | 68 | D |
| 0100 0101 | 45 | 69 | E |
| 0100 0110 | 46 | 70 | F |
| 0100 0111 | 47 | 71 | G |

| BINARY | HEX. | DEC. | COMMENTS |
|---|---|---|---|
| 0100 1000 | 48 | 72 | H |
| 0100 1001 | 49 | 73 | I |
| 0100 1010 | 4A | 74 | J |
| 0100 1011 | 4B | 75 | K |
| 0100 1100 | 4C | 76 | L |
| 0100 1101 | 4D | 77 | M |
| 0100 1110 | 4E | 78 | N |
| 0100 1111 | 4F | 79 | O |
| 0101 0000 | 50 | 80 | P |
| 0101 0001 | 51 | 81 | Q |
| 0101 0010 | 52 | 82 | R |
| 0101 0011 | 53 | 83 | S |
| 0101 0100 | 54 | 84 | T |
| 0101 0101 | 55 | 85 | U |
| 0101 0110 | 56 | 86 | V |
| 0101 0111 | 57 | 87 | W |
| 0101 1000 | 58 | 88 | X |
| 0101 1001 | 59 | 89 | Y |
| 0101 1010 | 5A | 90 | Z |
| 0101 1011 | 5B | 91 | UP ARROW CHAR. |
| 0101 1100 | 5C | 92 | DOWN ARROW CHAR. |
| 0101 1101 | 5D | 93 | LEFT ARROW CHAR. |
| 0101 1110 | 5E | 94 | RIGHT ARROW CHAR. |
| 0101 1111 | 5F | 95 | CURSOR CHAR. |
| 0110 0000 | 60 | 96 | (shifted @ key) |
| 0110 0001 | 61 | 97 | a |
| 0110 0010 | 62 | 98 | b |
| 0110 0011 | 63 | 99 | c |
| 0110 0100 | 64 | 100 | d |
| 0110 0101 | 65 | 101 | e |
| 0110 0110 | 66 | 102 | f |
| 0110 0111 | 67 | 103 | g |
| 0110 1000 | 68 | 104 | h |
| 0110 1001 | 69 | 105 | i |
| 0110 1010 | 6A | 106 | j |
| 0110 1011 | 6B | 107 | k |
| 0110 1100 | 6C | 108 | l |

| BINARY | HEX. | DEC. | COMMENTS |
|---|---|---|---|
| 0110 1101 | 6D | 109 | m |
| 0110 1110 | 6E | 110 | n |
| 0110 1111 | 6F | 111 | o |
| 0111 0000 | 70 | 112 | p |
| 0111 0001 | 71 | 113 | q |
| 0111 0010 | 72 | 114 | r |
| 0111 0011 | 73 | 115 | s |
| 0111 0100 | 74 | 116 | t |
| 0111 0101 | 75 | 117 | u |
| 0111 0110 | 76 | 118 | v |
| 0111 0111 | 77 | 119 | w |
| 0111 1000 | 78 | 120 | x |
| 0111 1001 | 79 | 121 | y |
| 0111 1010 | 7A | 122 | z |
| 0111 1011 | 7B | 123 | |
| 0111 1100 | 7C | 124 | |
| 0111 1101 | 7D | 125 | |
| 0111 1110 | 7E | 126 | |
| 0111 1111 | 7F | 127 | |
| 1000 0000 | 80 | 128 | END |
| 1000 0001 | 81 | 129 | FOR |
| 1000 0010 | 82 | 130 | RESET |
| 1000 0011 | 83 | 131 | SET |
| 1000 0100 | 84 | 132 | CLS |
| 1000 0101 | 85 | 133 | CMD |
| 1000 0110 | 86 | 134 | RANDOM |
| 1000 0111 | 87 | 135 | NEXT |
| 1000 1000 | 88 | 136 | DATA |
| 1000 1001 | 89 | 137 | INPUT |
| 1000 1010 | 8A | 138 | DIM |
| 1000 1011 | 8B | 139 | READ |
| 1000 1100 | 8C | 140 | LET |
| 1000 1101 | 8D | 141 | GOTO |
| 1000 1110 | 8E | 142 | RUN |
| 1000 1111 | 8F | 143 | IF |
| 1001 0000 | 90 | 144 | RESTORE |
| 1001 0001 | 91 | 145 | GOSUB |

- APPENDIX 1 -

| BINARY | HEX. | DEC. | COMMENTS |
|---|---|---|---|
| 1001 0010 | 92 | 146 | RETURN |
| 1001 0011 | 93 | 147 | REM |
| 1001 0100 | 94 | 148 | STOP |
| 1001 0101 | 95 | 149 | ELSE |
| 1001 0110 | 96 | 150 | TRON |
| 1001 0111 | 97 | 151 | TROFF |
| 1001 1000 | 98 | 152 | DEFSTR |
| 1001 1001 | 99 | 153 | DEFINT |
| 1001 1010 | 9A | 154 | DEFSNG |
| 1001 1011 | 9B | 155 | DEFDBL |
| 1001 1100 | 9C | 156 | LINE |
| 1001 1101 | 9D | 157 | EDIT |
| 1001 1110 | 9E | 158 | ERROR |
| 1001 1111 | 9F | 159 | RESUME |
| 1010 0000 | A0 | 160 | OUT |
| 1010 0001 | A1 | 161 | ON |
| 1010 0010 | A2 | 162 | OPEN |
| 1010 0011 | A3 | 163 | FIELD |
| 1010 0100 | A4 | 164 | GET |
| 1010 0101 | A5 | 165 | PUT |
| 1010 0110 | A6 | 166 | CLOSE |
| 1010 0111 | A7 | 167 | LOAD |
| 1010 1000 | A8 | 168 | MERGE |
| 1010 1001 | A9 | 169 | NAME |
| 1010 1010 | AA | 170 | KILL |
| 1010 1011 | AB | 171 | LSET |
| 1010 1100 | AC | 172 | RSET |
| 1010 1101 | AD | 173 | SAVE |
| 1010 1110 | AE | 174 | SYSTEM |
| 1010 1111 | AF | 175 | LPRINT |
| 1011 0000 | B0 | 176 | DEF |
| 1011 0001 | B1 | 177 | POKE |
| 1011 0010 | B2 | 178 | PRINT |
| 1011 0011 | B3 | 179 | CONT |
| 1011 0100 | B4 | 180 | LIST |
| 1011 0101 | B5 | 181 | LLIST |
| 1011 0110 | B6 | 182 | DELETE |

- APPENDIX 1 -

| BINARY | HEX. | DEC. | COMMENTS |
|---|---|---|---|
| 1011 0111 | B7 | 183 | AUTO |
| 1011 1000 | B8 | 184 | CLEAR |
| 1011 1001 | B9 | 185 | CLOAD |
| 1011 1010 | BA | 186 | CSAVE |
| 1011 1011 | BB | 187 | NEW |
| 1011 1100 | BC | 188 | TAB( |
| 1011 1101 | BD | 189 | TO |
| 1011 1110 | BE | 190 | FN |
| 1011 1111 | BF | 191 | USING |
| 1100 0000 | C0 | 192 | VARPTR |
| 1100 0001 | C1 | 193 | USR |
| 1100 0010 | C2 | 194 | ERL |
| 1100 0011 | C3 | 195 | ERR |
| 1100 0100 | C4 | 196 | STRING$ |
| 1100 0101 | C5 | 197 | INSTR |
| 1100 0110 | C6 | 198 | POINT |
| 1100 0111 | C7 | 199 | TIME$ |
| 1100 1000 | C8 | 200 | MEM |
| 1100 1001 | C9 | 201 | INKEY$ |
| 1100 1010 | CA | 202 | THEN |
| 1100 1011 | CB | 203 | NOT |
| 1100 1100 | CC | 204 | STEP |
| 1100 1101 | CD | 205 | + |
| 1100 1110 | CE | 206 | - |
| 1100 1111 | CF | 207 | * |
| 1101 0000 | D0 | 208 | / |
| 1101 0001 | D1 | 209 | EXP. SIGN (up arrow, ESC) |
| 1101 0010 | D2 | 210 | AND |
| 1101 0011 | D3 | 211 | OR |
| 1101 0100 | D4 | 212 | GREATER THAN |
| 1101 0101 | D5 | 213 | = |
| 1101 0110 | D6 | 214 | LESS THAN |
| 1101 0111 | D7 | 215 | SGN |
| 1101 1000 | D8 | 216 | INT |
| 1101 1001 | D9 | 217 | ABS |
| 1101 1010 | DA | 218 | FRE |
| 1101 1011 | DB | 219 | INP |

| BINARY | HEX. | DEC. | COMMENTS |
|---|---|---|---|
| 1101 1100 | DC | 220 | POS |
| 1101 1101 | DD | 221 | SQR |
| 1101 1110 | DE | 222 | RND |
| 1101 1111 | DF | 223 | LOG |
| 1110 0000 | E0 | 224 | EXP |
| 1110 0001 | E1 | 225 | COS |
| 1110 0010 | E2 | 226 | SIN |
| 1110 0011 | E3 | 227 | TAN |
| 1110 0100 | E4 | 228 | ATN |
| 1110 0101 | E5 | 229 | PEEK |
| 1110 0110 | E6 | 230 | CVI |
| 1110 0111 | E7 | 231 | CVS |
| 1110 1000 | E8 | 232 | CVD |
| 1110 1001 | E9 | 233 | EOF |
| 1110 1010 | EA | 234 | LOC |
| 1110 1011 | EB | 235 | LOF |
| 1110 1100 | EC | 236 | MKI$ |
| 1110 1101 | ED | 237 | MKS$ |
| 1110 1110 | EE | 238 | MKD$ |
| 1110 1111 | EF | 239 | CINT |
| 1111 0000 | F0 | 240 | CSNG |
| 1111 0001 | F1 | 241 | CDBL |
| 1111 0010 | F2 | 242 | FIX |
| 1111 0011 | F3 | 243 | LEN |
| 1111 0100 | F4 | 244 | STR$ |
| 1111 0101 | F5 | 245 | VAL |
| 1111 0110 | F6 | 246 | ASC |
| 1111 0111 | F7 | 247 | CHR$ |
| 1111 1000 | F8 | 248 | LEFT$ |
| 1111 1001 | F9 | 249 | RIGHT$ |
| 1111 1010 | FA | 250 | MID$ |
| 1111 1011 | FB | 251 | (Single Quote char - REM) |
| 1111 1100 | FC | 252 | |
| 1111 1101 | FD | 253 | |
| 1111 1110 | FE | 254 | |
| 1111 1111 | FF | 255 | |

## --** APPENDIX 2 **--

### SAMPLE PROGRAM NO. 1

The program listed below shows how the DOS link addresses can be used. It provides a new BASIC command, the syntax of which is CMD"B,X". Where "X" is any BASIC variable. When this command is executed, the binary value of the variable specified will be printed to the screen. Note that the binary value returned is in twos complement form. Since the routine is going to be part of the BASIC interpreter, it is reasonable to put it in low memory and move the BASIC program pointers past it. This is achieved by the routine labeled INIT. This part of the routine zeroes three bytes in memory where the start of BASIC memory is going to be and fixes the start of BASIC pointer, variables pointer, the array pointer and the free space pointer accordingly. Finally, the CMD link address is loaded with the entry point to the main program. This program finds the value of a variable and displays it as a 16 bit binary value to the screen. The numbers must fall in the range of a BASIC integer value.

Although the program is not terribly useful, it does demonstrate the principles involved. Line 170 shows the use of a CALL 2337H which evaluates the current BASIC expression, checks for errors and places the string part (the part between quotes) of the CMD command in the string work area and loads the buffer with the string length and string address. The call 29D7H does some housekeeping for BASIC and makes the HL register pair point to the first of the three bytes containing string length followed by string address. Line 200 to 220 checks if the string length is zero and generates an ILLEGAL FUNCTION CALL ERROR if it is. Line 240 to 270 gets the string address and checks if the

--** APPENDIX 2 **--

SAMPLE PROGRAM

(continued)

control character is a "B". The rest of the program is straight forward, the comments show what it does. The best way to learn how to use the ROM routines is by experimenting with them so try some and see their effect, use a monitor if necessary to examine memory areas such as the ACC area and pointers.

Try the new command in the following way:

```
10 FOR N = 0 TO 15: CMD"B,N": PRINT: NEXT N
```

This will print a column of binary numbers from 0 to 15.

```
0010        ORG     433FH
0020 INIT   LD      HL,FINISH+1
0030        LD      (40A4H),HL ;LOAD NEW START OF BASIC
0040                           ; ADDRESS.
0050        CALL    1B4DH      ;INIT ALL OTHER POINTERS
0060        LD      HL,ENTRY   ;GET ENTRY ADDRESS
0070        LD      (4174H),HL ;FIX CMD LINK VECTOR
0080        JP      06CCH      ;RETURN TO BASIC
0090 ENTRY  CALL    2337H      ;EVALUATE EXPRESSION
0100        PUSH    HL
0110        CALL    29D7H      ;GET STRING ADDRESS ETC
0120        LD      A,(HL)     ;GET STRING LENGTH
0130        OR      A          ;SET FLAGS
0140        JP      Z,1E4AH    ;IF ZERO FC ERROR
0150        INC     HL
0160        LD      E,(HL)
0170        EX      DE,HL      ;STRING ADDRESS IN HL
0180        LD      A,(HL)     ;GET FIRST CHARACTER
0190        CP      "B"        ;IS IT B ?
0200        JP      NZ,1E4AH   ;ONLY ACCEPT CMD"B"
0210        RST     10H        ;FIND NEXT CHARACTER
0220        RST     8H         ;IS IT A COMMA ?
0230        DEFB    ","
0240        CALL    2540H      ;PUT VALUE OF VAR IN ACC
0250        CALL    0A7FH      ;CONVERT TO INTEGER
0260        CALL    BINOUT     ;DISPLAY MSB BYTE
0270        LD      H,L
0280        CALL    BINOUT     ;DISPLAY LSB BYTE
0290        POP     HL
0300        RET                ;DONE !
0310 BINOUT LD      B,8        ;8 BITS IN BYTE
0320 LOOP   XOR     A          ;CLEAR A
0330        SLA     H          ;SHIFT BIT INTO C FLAG
0340        ADC     A,30H      ;ADD BIT AND CONVERT TO
0350                           ; ASCII
0360        CALL    33H        ;DISPLAY IT!
0370        DJNZ    LOOP       ;LOOP TILL 8 BITS DONE
0380        LD      A,20H      ;OUTPUT SPACE
0390        CALL    33H
```

```
0400 FINISH DEFW   0          ;NEW START OF BASIC
0405                          ;LOCATION
0410        DEFW   0
0420        END    INIT
```

LOCATION DESCRIPTION

## --** APPENDIX 3 **--

## - SAMPLE PROGRAM NO.2 -

This program shows how to manipulate the BASIC pointers and how to append code to an existing BASIC program in memory. The program takes a block of memory and creates a BASIC program containing DATA statements and a FOR-NEXT loop to READ the DATA and POKE it back into memory. This is a rather useful program as often it is the only way to include machine code in a BASIC program. The routine will automatically append the new BASIC code to the end of an existing program or create a new BASIC program if none is resident. The routine uses quite a few ROM routines which are defined in the LABEL DEFENITION table. This table can be consulted to see which routines are used in this program. The ROM reference section can then be consulted to see how the ROM call was applied. Finally the source code listing was assembled and created with the MACRO-80 assembler using a MODEL II computer. That is why the labels have colons behind them, if EDTASM is used don°t put these colons in. The pseudo-ops: PAGE, TITLE, SUBTTL and COMMENT should not be used by the EDTASM user as they are not available, these are only used to get an easy to read listing.

The program is accessed by issuing the NAME command. As soon as NAME is entered, the screen is cleared and the question "START and END addresses in HEX (START,END) ?" appears. This should be answered with two four digit hexadecimal addresses separated with a comma. When the "READY" prompt re-apears a new BASIC program will be in memory, or it will be appended to an existing program, this can be verified by LISTing it.

```
00010          TITLE   DATGEN Data generator version 1.00 for TRS-80
00020          .COMMENT!
00030
00040   ##########################################
00050   ##                                      ##
00060   ##  DATGEN version 1.00 for TRS-80 M1&3 ##
00070   ##  Created     :       25/04/1982      ##
00080   ##  Programmer  :       E. R. Paay      ##
00090   ##  Copyright   :       InterSoft (C)   ##
00100   ##                                      ##
00110   ##########################################
00120
00130   This program is written to demonstrate the TRS-80 ROM calls.
00140   Any given memory block can be converted to a series of DATA
00150   statements and a FOR-NEXT loop to READ the DATA statements
00160   and POKE their values back into their corresponding memory
00170   addresses by this program.
00180   !
00190          PAGE    29
```

```
                        00200  ;       --** LABEL DEFINITIONS **--
                        00210  ;
1AF8                    00220  REPAIR  EQU     1AF8H   ; Check and repair the BASIC line-pointers
0FBD                    00230  HEXDEC  EQU     0FBDH   ; Hex. to Dec conversion
0032                    00240  LNLEN   EQU     0050    ; Line length to use
0A9A                    00250  PUTACC  EQU     0A9AH   ; Put HL in ACC
0088                    00260  DATA    EQU     0088H   ; 88H is compression code for DATA
28A7                    00270  PRINT   EQU     28A7H   ; Print routine
409C                    00280  PRTFLG  EQU     409CH   ; I/O device flag
05D9                    00290  INPUT   EQU     05D9H   ; INPUT routine
0010                    00300  FNDCHR  EQU     0010H   ; Find non blank characters
40A7                    00310  BUFPTR  EQU     40A7H   ; I/O Buffer pointer
0008                    00320  CHKCHR  EQU     0008H   ; Check next character
40A4                    00330  STBAS   EQU     40A4H   ; Start of BASIC pointer
40F9                    00340  ENDBAS  EQU     40F9H   ; End of BASIC pointer
1A19                    00350  RETBAS  EQU     1A19H   ; Return to BASIC (Model 1 use 06CCH)
418E                    00360  NAME    EQU     418EH   ; Use name command to enter routine
00C3                    00370  JMP     EQU     00C3H   ; Opcode for JP
01C9                    00380  CLS     EQU     01C9H   ; CLS routine
000A                    00390  BUFLEN  EQU     0010    ; Buffer length
00CE                    00400  MINUS   EQU     00CEH   ; Compression code for "-"
1B6E                    00410  RESPTR  EQU     1B6EH   ; Reset BASIC pointers
0005                    00420  MINLNO  EQU     0005    ; Minimum line number value
```

```
0002                    00430  LNFSET  EQU     0002    ; Offset for NEW linenumber
                        00440
                        00450          SUBTTL  Initialization section
                        00460          PAGE
```

```
                        00470  ;       --**    INITIALIZATION SECTION **--
                        00480
                        00490  ;       --** DON'T FORGET TO SET MEMORY SIZE **--
                        00500
                        00510          ORG     7E00H           ; Put your ORiGin here
7E00'  3E C3            00520  INIT:   LD      A,JMP           ; Load code for JumP in A
7E02'  21 7E0E'         00530          LD      HL,START        ; Load our entry point in HL
7E05'  32 418E          00540          LD      (NAME),A        ; Initialize the NAME vector
7E08'  22 418F          00550          LD      (NAME+1),HL
7E0B'  C3 1A19          00560          JP      RETBAS          ; Initialization done!
                        00570          SUBTTL  Main program area
                        00580          PAGE
```

```
                        00590  ;       --** MAIN PROGRAM SECTION **--
                        00600  ;
7E0E'  31 7DFE'         00610  START:  LD      SP,INIT-2       ; Set stackpointer
7E11'  CD 01C9          00620          CALL    CLS             ; Clear the screen
7E14'  AF               00630          XOR     A               ; Clear the A register
7E15'  32 409C          00640          LD      (PRTFLG),A      ; Set I/O flag to CRT output
7E18'  21 7F94'         00650          LD      HL,STMSGE       ; Point HL to start of text
7E1B'  CD 28A7          00660          CALL    PRINT           ; PRINT question
7E1E'  06 0A            00670          LD      B,BUFLEN        ; Limit input to 10 char's max.
7E20'  21 7F8A'         00680          LD      HL,BUFR         ; Point HL to INPUT buffer
7E23'  CD 05D9          00690          CALL    INPUT           ; INPUT operator responce
7E26'  2B               00700          DEC     HL              ; Adjust pointer for RST 10H
7E27'  D7               00710          RST     FNDCHR          ; Find first non blank char.
7E28'  CD 7F21'         00720          CALL    ASCHEX          ; Convert START address to binary
7E2B'  ED 53 7F88'      00730          LD      (STMEM),DE      ; Save address to start from
7E2F'  CF               00740          RST     CHKCHR          ; Check if next char. is a ","
7E30'  2C               00750          DEFB    ','             ; Place a comma here for RST 8
7E31'  CD 7F21'         00760          CALL    ASCHEX          ; Convert END address to binary
7E34'  13               00770          INC     DE              ; Give it one extra
7E35'  ED 53 7F86'      00780          LD      (ENDMEM),DE     ; Save end address
7E39'  B7               00790          OR      A               ; Make sure CY flag is reset
7E3A'  2A 40F9          00800          LD      HL,(ENDBAS)     ; Check if a BASIC program is
7E3D'  ED 5B 40A4       00810          LD      DE,(STBAS)      ; - already resident
```

```
7E41'  ED 52          00820          SBC    HL,DE
7E43'  D5             00830          PUSH   DE            ; Save DE
7E44'  11 0004        00840          LD     DE,4           ; If difference between START and
                      00850                                ; END pointers is less than 4 no
                      00860                                ; BASIC program is resident
7E47'  B7             00870          OR     A              ; Make sure CY flag is reset
7E48'  ED 52          00880          SBC    HL,DE
7E4A'  D1             00890          POP    DE             ; Restore DE
7E4B'  30 0C          00900          JR     NC,SETPTR      ; Jump to SETPTR if prog. resident
                      00910
                      00920  ;       --** Continues here if NO BASIC program resident **--
                      00930
7E4D'  21 0005        00940          LD     HL,MINLNO      ; Set minimum linenumber value to use
7E50'  22 7F82'       00950          LD     (LNUM),HL
7E53'  D5             00960          PUSH   DE
7E54'  E1             00970          POP    HL             ; Duplicate DE into HL
7E55'  E5             00980  CNTO:   PUSH   HL
7E56'  D1             00990          POP    DE
7E57'  18 21          01000          JR     BUILD          ; Start to build BASIC line
                      01010
                      01020  ;       --** Continues here if a BASIC program IS resident **--
                      01030
7E59'  2A 40A4        01040  SETPTR: LD     HL,(STBAS)     ; First find HIGEST line num.
7E5C'  23             01050          INC    HL
7E5D'  23             01060  NEXTLN: INC    HL             ; Bypass line pointer
```

```
7E5E' 5E          01070          LD    E,(HL)        ; Get line number into DE
7E5F' 23          01080          INC   HL
7E60' 56          01090          LD    D,(HL)
7E61' 23          01100          INC   HL
7E62' 7E          01110  FNDEND: LD    A,(HL)        ; Find end of line
7E63' 23          01120          INC   HL
7E64' B7          01130          OR    A
7E65' 20 FB       01140          JR    NZ,FNDEND     ; Find ZERO delimiter byte
                  01150                              ; - which signals END of line
                  01160
                  01170  ;       --** Continues here if END of program line found **--
                  01180
7E67' 7E          01190          LD    A,(HL)
7E68' 23          01200          INC   HL            ; Now check for a further 2
7E69' B6          01210          OR    (HL)          ; - ZERO bytes which signal
                  01220                              ; - the end of program
7E6A' 20 F1       01230          JR    NZ,NEXTLN     ; If not end-of-prog try next line
7E6C' 21 0002     01240          LD    HL,LNFSET     ; Load offset and calculate new
7E6F' 19          01250          ADD   HL,DE         ; - line number
7E70' 22 7F82'    01260          LD    (LNUM),HL     ; Store new line number
7E73' 2A 40F9     01270          LD    HL,(ENDBAS)   ; Compute where to start our
7E76' 2B          01280          DEC   HL            ; - BASIC program
7E77' 2B          01290          DEC   HL
7E78' 18 DB       01300          JR    CNTO
                  01310
```

```
                        01320  ;       --** Continues here when all parameters have been    **--
                        01330  ;       --** computed to allow us to "BUILD" a BASIC program **--
                        01340
7E7A'  2A 7F88'         01350  BUILD:  LD      HL,(STMEM)      ; Point to memory block
7E7D'  06 32            01360  BNXTLN: LD      B,LNLEN         ; Maximum NO of DATA items
7E7F'  3E FF            01370          LD      A,0FFH          ; Put filler in temp. linepointer
7E81'  12               01380          LD      (DE),A
7E82'  13               01390          INC     DE
7E83'  12               01400          LD      (DE),A
7E84'  13               01410          INC     DE
7E85'  E5               01420          PUSH    HL              ; Save HL
7E86'  2A 7F82'         01430          LD      HL,(LNUM)       ; Get OLD linenumber
7E89'  23               01440          INC     HL              ; Calculate NEW
7E8A'  22 7F82'         01450          LD      (LNUM),HL
7E8D'  EB               01460          EX      DE,HL
7E8E'  73               01470          LD      (HL),E          ; INSERT linenumber
7E8F'  23               01480          INC     HL
7E90'  72               01490          LD      (HL),D
7E91'  23               01500          INC     HL
7E92'  3E 88            01510          LD      A,DATA          ; INSERT code for "DATA"
7E94'  77               01520          LD      (HL),A
7E95'  23               01530          INC     HL
7E96'  EB               01540          EX      DE,HL
7E97'  E1               01550          POP     HL              ; Restore HL
7E98'  7E               01560  GETBYT: LD      A,(HL)          ; Get a byte from memory block
```

```
7E99'  23              01570          INC     HL
7E9A'  E5              01580          PUSH    HL              ; Save memory pointer
7E9B'  6F              01590          LD      L,A
7E9C'  26 00           01600          LD      H,0             ; Prepare for Binary to Decimal
                       01610                                  ; - conversion
7E9E'  CD 0A9A         01620          CALL    PUTACC          ; Put value in ACC
7EA1'  CD 7F7A'        01630          CALL    HEXASC          ; Convert BIN. to ASCII decimal
7EA4'  2B              01640          DEC     HL              ; HL points to ASCII string
7EA5'  D7              01650          RST     FNDCHR          ; Find first character
7EA6'  7E              01660  LOOP1:  LD      A,(HL)          ; Get a character
7EA7'  23              01670          INC     HL
7EA8'  B7              01680          OR      A
7EA9'  28 04           01690          JR      Z,EXIT1         ; Delimiter ?
7EAB'  12              01700          LD      (DE),A          ; Insert data in program line
7EAC'  13              01710          INC     DE
7EAD'  18 F7           01720          JR      LOOP1           ; Loop till done
7EAF'  E1              01730  EXIT1:  POP     HL              ; Get memory pointer
7EB0'  E5              01740          PUSH    HL
7EB1'  78              01750          LD      A,B
7EB2'  ED 4B 7F86'     01760          LD      BC,(ENDMEM)
7EB6'  B7              01770          OR      A               ; Reset CY flag
7EB7'  ED 42           01780          SBC     HL,BC
7EB9'  47              01790          LD      B,A
7EBA'  E1              01800          POP     HL
7EBB'  30 0C           01810          JR      NC,FINISH       ; Check if all memory block done
```

```
7EBD'  3E 2C          01820          LD    A,','        ; Separate DATA entry's with a comma
7EBF'  12             01830          LD    (DE),A       ; Insert comma
7EC0'  13             01840          INC   DE
7EC1'  10 D5          01850          DJNZ  GETBYT       ; Loop to fill 1 program line
7EC3'  AF             01860          XOR   A
7EC4'  1B             01870          DEC   DE           ; Erase last comma
7EC5'  12             01880          LD    (DE),A       ; Insert END of line delimiter
7EC6'  13             01890          INC   DE
7EC7'  18 B4          01900          JR    BNXTLN       ; "BUILD" next line
7EC9'  AF             01910  FINISH: XOR   A
7ECA'  12             01920          LD    (DE),A       ; Insert end of line delimiter
7ECB'  13             01930          INC   DE
7ECC'  3C             01940          INC   A
7ECD'  12             01950          LD    (DE),A       ; Insert dummy line-pointer
7ECE'  13             01960          INC   DE
7ECF'  12             01970          LD    (DE),A
7ED0'  13             01980          INC   DE
7ED1'  2A 7F82'       01990          LD    HL,(LNUM)    ; Get next line number
7ED4'  23             02000          INC   HL
7ED5'  23             02010          INC   HL
7ED6'  EB             02020          EX    DE,HL
7ED7'  73             02030          LD    (HL),E       ; Insert linenumber
7ED8'  23             02040          INC   HL
7ED9'  72             02050          LD    (HL),D
7EDA'  23             02060          INC   HL
```

```
7EDB'  EB              02070          EX     DE,HL
7EDC'  2A 7F88'        02080          LD     HL,(STMEM)     ; Get start of memory block address
7EDF'  CD 0A9A         02090          CALL   PUTACC         ; Put it in ACC
7EE2'  CD 7F7A'        02100          CALL   HEXASC
7EE5'  D5              02110          PUSH   DE
7EE6'  11 7FCE'        02120          LD     DE,SRTVAL      ; Point DE to model BASIC line
7EE9'  CD 7F61'        02130          CALL   INSERT         ; Insert START address into FOR-NEXT
                       02140                                ; - loop
7EEC'  2A 7F86'        02150          LD     HL,(ENDMEM)    ; Get end of memory block address
7EEF'  2B              02160          DEC    HL
7EF0'  CD 0A9A         02170          CALL   PUTACC         ; Put it in ACC
7EF3'  CD 7F7A'        02180          CALL   HEXASC         ; Convert it to ASCII decimal string
7EF6'  11 7FD5'        02190          LD     DE,ENDVAL      ; Point DE to FOR/NEXT loop
7EF9'  CD 7F61'        02200          CALL   INSERT         ; Insert secon value in FOR/NEXT loop
7EFC'  D1              02210          POP    DE
7EFD'  21 7FCA'        02220          LD     HL,PROGRM      ; Point HL to model BASIC line
7F00'  7E              02230  LOOP3:  LD     A,(HL)         ; Put model BASIC line in our
                       02240                                ; - actual BASIC program
7F01'  23              02250          INC    HL
7F02'  B7              02260          OR     A
7F03'  28 08           02270          JR     Z,DONE
7F05'  FE 20           02280          CP     ' '            ; Ignore spaces
7F07'  28 F7           02290          JR     Z,LOOP3
7F09'  12              02300          LD     (DE),A
7F0A'  13              02310          INC    DE
```

```
7F0B'  18 F3           02320          JR     LOOP3
7F0D'  AF              02330  DONE:   XOR    A
7F0E'  06 03           02340          LD     B,3
7F10'  12              02350  LOOP4:  LD     (DE),A          ; Insert END-OF-PROGRAM marker
7F11'  13              02360          INC    DE
7F12'  10 FC           02370          DJNZ   LOOP4
7F14'  ED 53 40F9      02380          LD     (ENDBAS),DE     ; Adjust BASIC pointers
7F18'  CD 1B6E         02390          CALL   RESPTR          ; Reset reserved RAM pointers
7F1B'  CD 1AF8         02400          CALL   REPAIR          ; Adjust dummy linepointers to
                       02410                                 ; - their real values
                       02420
                       02430  ;       --** ALL DONE !!! - We have now created a BASIC   **--
                       02440  ;       --** program that will read back a block of memory**--
                       02450
7F1E'  C3 1A19         02460          JP     RETBAS          ; Exit to BASIC READY mode
                       02470          SUBTTL Subroutine section
                       02480          PAGE
```

```
                        02490  ;       --** SUBROUTINE SECTION **--
                        02500
7F21'  CD 7F2A'         02510  ASCHEX: CALL    CONBYT          ; Convert ASCII-Hex. to BINARY
7F24'  57               02520          LD      D,A             ; - result is in DE
7F25'  CD 7F2A'         02530          CALL    CONBYT
7F28'  5F               02540          LD      E,A
7F29'  C9               02550          RET
                        02560
7F2A'  CD 7F39'         02570  CONBYT: CALL    FETCH           ; Get an ASCII character
7F2D'  CD 7F50'         02580          CALL    CONV
7F30'  5F               02590          LD      E,A
7F31'  CD 7F39'         02600          CALL    FETCH
7F34'  CD 7F54'         02610          CALL    CONV1
7F37'  B3               02620          OR      E
7F38'  C9               02630          RET
                        02640
7F39'  7E               02650  FETCH:  LD      A,(HL)
7F3A'  23               02660          INC     HL              ; Get char
7F3B'  FE 30            02670          CP      '0'             ; Allow only VALID hex. characters
7F3D'  38 0E            02680          JR      C,ERROR
7F3F'  FE 3A            02690          CP      ':'
7F41'  30 01            02700          JR      NC,FT1
7F43'  C9               02710          RET
```

```
7F44'  FE 41          02720  FT1:    CP     'A'
7F46'  38 05          02730         JR     C,ERROR
7F48'  FE 47          02740         CP     'G'
7F4A'  30 01          02750         JR     NC,ERROR
7F4C'  C9             02760         RET
                      02770
7F4D'  C3 7E0E'       02780  ERROR:  JP     START          ; If BAD entry start all over again
                      02790
7F50'  01 7F5C'       02800  CONV:   LD     BC,SHIFT
7F53'  C5             02810         PUSH   BC             ; Put new RETurn address on stack
7F54'  D6 30          02820  CONV1:  SUB    30H
7F56'  FE 0A          02830         CP     0AH
7F58'  D8             02840         RET    C
7F59'  D6 07          02850         SUB    7
7F5B'  C9             02860         RET
                      02870
7F5C'  0F             02880  SHIFT:  RRCA
7F5D'  0F             02890         RRCA
7F5E'  0F             02900         RRCA
7F5F'  0F             02910         RRCA
7F60'  C9             02920         RET
                      02930
7F61'  06 06          02940  INSERT: LD     B,6
7F63'  7E             02950  LOOP2:  LD     A,(HL)         ; Insert 6 characters
7F64'  23             02960         INC    HL
```

Subroutine section

```
7F65'  B7              02970          OR     A
7F66'  28 0B           02980          JR     Z,FILLER
7F68'  FE 2D           02990          CP     '-'            ; Negative number ?
7F6A'  20 02           03000          JR     NZ,CNT1
7F6C'  3E CE           03010          LD     A,MINUS        ; If negative put correct BASIC code
7F6E'  12              03020  CNT1:   LD     (DE),A
7F6F'  13              03030          INC    DE
7F70'  10 F1           03040          DJNZ   LOOP2
7F72'  C9              03050          RET
7F73'  3E 20           03060  FILLER: LD     A,' '          ; Pad with blanks
7F75'  12              03070          LD     (DE),A
7F76'  13              03080          INC    DE
7F77'  10 FA           03090          DJNZ   FILLER
7F79'  C9              03100          RET
                       03110
7F7A'  C5              03120  HEXASC: PUSH   BC
7F7B'  D5              03130          PUSH   DE
7F7C'  CD 0FBD         03140          CALL   HEXDEC
7F7F'  D1              03150          POP    DE
7F80'  C1              03160          POP    BC
7F81'  C9              03170          RET
                       03180
                       03190          SUBTTL Data buffer and storage section
                       03200          PAGE
```

Data buffer and storage section

```
                                03210  ;       --** DATA STORAGE SECTION **--
                                03220
7F82'  0000                     03230  LNUM:   DEFW    0
7F84'  0000                     03240  LPOINT: DEFW    0
7F86'  0000                     03250  ENDMEM: DEFW    0
7F88'  0000                     03260  STMEM:  DEFW    0
7F8A'                           03270  BUFR:   DEFS    BUFLEN
7F94'  09 53 54 41              03280  STMSGE: DEFM    '       START and END addresses in HEX '
7F98'  52 54 20 61
7F9C'  6E 64 20 45
7FA0'  4E 44 20 61
7FA4'  64 64 72 65
7FA8'  73 73 65 73
7FAC'  20 69 6E 20
7FB0'  48 45 58 20
7FB4'  28 53 54 41              03290          DEFM    '(START,END) ?'
7FB8'  52 54 2C 45
7FBC'  4E 44 29 20
7FC0'  3F
7FC1'  0D0D                     03300          DEFW    0D0DH
7FC3'  20 20 20 20              03310          DEFM    '       '       ; 6 SPACES
7FC7'  20 20
7FC9'  00                       03320          DEFB    0
```

Data buffer and storage section

```
7FCA'  81                  03330  PROGRM: DEFB  81H
7FCB'  58 58               03340          DEFM  'XX'
7FCD'  D5                  03350          DEFB  0D5H
7FCE'  20 20 20 20         03360  SRTVAL: DEFM  '      '
7FD2'  20 20
7FD4'  BD                  03370          DEFB  0BDH
7FD5'  20 20 20 20         03380  ENDVAL: DEFM  '      :'
7FD9'  20 20 3A
7FDC'  8B                  03390          DEFB  8BH
7FDD'  59 59 3A            03400          DEFM  'YY:'
7FE0'  B1                  03410          DEFB  0B1H
7FE1'  58 58 2C 59         03420          DEFM  'XX,YY:'
7FE5'  59 3A
7FE7'  87                  03430          DEFB  87H
7FE8'  58 58               03440          DEFM  'XX'
7FEA'  00                  03450          DEFB  0
                           03460          END   INIT
```

Data buffer and storage section

Macros:

Symbols:

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| ASCHEX | 7F21' | BNXTLN | 7E7D' | BUFLEN | 000A | BUFPTR | 40A7 |
| BUFR | 7F8A' | BUILD | 7E7A' | CHKCHR | 0008 | CLS | 01C9 |
| CNT0 | 7E55' | CNT1 | 7F6E' | CONBYT | 7F2A' | CONV | 7F50' |
| CONV1 | 7F54' | DATA | 0088 | DONE | 7F0D' | ENDBAS | 40F9 |
| ENDMEM | 7F86' | ENDVAL | 7FD5' | ERROR | 7F4D' | EXIT1 | 7EAF' |
| FETCH | 7F39' | FILLER | 7F73' | FINISH | 7EC9' | FNDCHR | 0010 |
| FNDEND | 7E62' | FT1 | 7F44' | GETBYT | 7E98' | HEXASC | 7F7A' |
| HEXDEC | 0FBD | INIT | 7E00' | INPUT | 05D9 | INSERT | 7F61' |
| JMP | 00C3 | LNFSET | 0002 | LNLEN | 0032 | LNUM | 7F82' |
| LOOP1 | 7EA6' | LOOP2 | 7F63' | LOOP3 | 7F00' | LOOP4 | 7F10' |
| LPOINT | 7F84' | MINLND | 0005 | MINUS | 00CE | NAME | 418E |
| NEXTLN | 7E5D' | PRINT | 28A7 | PROGRM | 7FCA' | PRTFLG | 409C |
| PUTACC | 0A9A | REPAIR | 1AF8 | RESPTR | 1B6E | RETBAS | 1A19 |
| SETPTR | 7E59' | SHIFT | 7F5C' | SRTVAL | 7FCE' | START | 7E0E' |
| STBAS | 40A4 | STMEM | 7F88' | STMSGE | 7F94' | | |

No  Fatal error(s)

**** PART 3 ****

DBUG VERSION 1.00 FOR THE TRS-80 MODEL I or III

Feb 8, 1982

BY EDWIN R. PAAY.

(C) COPYRIGHT INTERSOFT, ALL RIGHTS RESERVED.

**** IMPORTANT NOTICE ****

InterSoft or its distributors shall have no liability or responsibility to any person or entity with respect to any liability, loss or damage caused or alleged to be caused directly or indirectly by programs sold by InterSoft or its distributors including but not limited to any interruption of service, loss of business or anticipatory profits or consequencial damages resulting from the use or operation of such computer programs.

DISTRIBUTED BY MICRO-80 PRODUCTS
433 MORPHETT STREET, ADELAIDE 5000
PH. (08) 211 7244.

(1). TRSDOS is a trademark of the Tandy corporation.
(2). ZBUG is produced by Microsoft.

## -FOREWORD-

This utility program is provided to make this a complete ASSEMBLY LANGUAGE package for the TRS-80. The ROM REFERENCE section describes where USEFUL subroutine calls are located and the DBUG program allows us to debug machine language programs on the TRS-80. This allows the user to try the ROM subroutines set out in the ROM reference manual while in debug mode and actually see registers, memory and/or screen change in response to the subroutines. I feel that the DBUG utility will make writing and debugging assembly language programs much simpler than ever before. I hope that you will feel the same. Happy debugging !

Edwin R. Paay

## *** INDEX ***



******

# * DBUG - SYMBOLIC DEBUGGER AND SINGLE STEP PROCESSOR *

## ** INTRODUCTION **

DBUG has been created for the assembly language programmer to make debugging assembly language programs much easier. Often programs can contain faults which can only be found by single stepping through them, while observing the registers and memory locations after each step. DBUG allows the user to do this and much more. The TRSDOS (1) DEBUG fills the entire screen with hexdumps over-writing all that was on the screen and, above all, it does not disassemble the current OP-code to which the PC register points. Other debuggers such as ZBUG (2) are powerful but have a range of commands so hard to memorize that continual referal to the manual is required for command syntax. Also, ZBUG falls short by not showing the register contents after each step unless the user asks to see them, this makes using ZBUG tedious.

The experienced assembly language programmer will find a flexibility never experienced before by the MODEL I or III user. The beginner assembly language programmer will find in DBUG an ease of use not offered by any other debugger. Often the beginner is turned away from assembly language programming because of the difficulty in debugging his programs. A fault as simple as leaving out a POP before executing a RET for instance could cause an intermittent fault with the program responding differently every time it is executed. These sort of faults can be very hard to find but will show up easily by single stepping. This is why many would-be assembly language programmers turn to so called "high level" languages like BASIC and never get to know the power of assembly language. I am convinced that if a good assembler and debugging utility like DBUG is used that writing assembly language programs can often be just as easy as writing

a program in a "high level language". Even more so if the ROM calls described in the ROM REFERENCE manual are used.

DBUG was written with the following aims in mind:

1). The work display should use as little space on the screen as possible.

2). Each location pointed to by the PC should be shown in HEX and in MNEMONICS.

3). All registers, memory locations etc. should be alterable by the user.

4). Break point handling should be allowed for.

5). The user should be able to single step through a program or execute CALLs and RSTs at choice.

Finally I would like to explain the difference between a monitor and a debugger. The difference between these two utilities is not well defined, but basically, a monitor is a program which can display HEX dumps and ASCII dumps of memory areas and can be used to examine and edit memory, move blocks of data to and from disk, zero memory and search for bytes or words in memory, to mention just a few common functions. A debugger must, as its most basic function, allow the user to single step through a program and dynamically show the programmer the effects of each step. Everything else that the debugger does centres around this main function. When you first look at the work display of DBUG you might wonder why it only uses three lines at the bottom of the screen. This means for instance that only 8 bytes are shown at any time by the memory display section while in HEX mode. This has been

done to keep as much of the screen free as possible for use by your application programs which may use the screen to display text and/or graphics. If DBUG used more display area text written by your programs would most likely end up being over written by hex. dumps and the like. The idea here is that the DBUG work display is a "window" into the Z80 cpu, which shows the status and contents of the registers at all times. This window should be as small as possible so as not to interfere with anything else.

## **** THE WORK DISPLAY ****

As was said earlier the work display is a window into the Z80 cpu and memory. This section will discuss the format used.

The work display uses the bottom 4 lines on the screen, a typical display might look like this:

```
A F     BC   DE   HL   IX   IY   PC ) ( RST 30        )
00 ZN   02B2 187F 614F 2208 FFFF 1350) F719 2221 41D1 E170
2800) 41D7 CD2C 25E5 2190    SP= FFBF) 1E1D 1E1D EF19 0000
```

On the top line the general purpose registers are shown with their contents underneath them on the second line. In the right top corner, the current disassembled opcode is shown between brackets. The F (flag) register is shown with its flags, if the flag is displayed it is set (1) if it is not shown it is reset (0). In the example the PC register has 1350 underneath it. This means that the PC register contains 1350H, the "]" following the 1350 signifies that this section of memory is displayed to the right of it, so that we can see where the PC register is pointing. On the third line 2800] is displayed, this is the memory pointer, the "]" signifies that the data following it is the contents of the memory location displayed. (NOTE that on your screen and in the sample above the "]" is displayed as the "greater-than" symbol) The same line shows the SP (stack pointer) in the same fashion. The fourth line is used for command input.

## *** DBUG COMMAND LIST. ***

All commands are entered from the keyboard with their parameters, if specified. In the command decription below nnnn represents a four digit HEXADECIMAL number. Incorrect entries will cause an error message to appear. All commands shown with spaces between them can be used with or without these spaces. The commands shown in lowercase are input using the SHIFT key. (note that on the model 3 a shift 0 must be used before these commands can be output) All commands must be followed with the ENTER key to execute them. This allows an erronous entry to be edited before it is executed.

| COMMAND | DESCRIPTION |
|---|---|
| A | (Ascii) This command causes all data to be displayed in ASCII format. The most significant bit is stripped before the characters are displayed. |
| B nnnn | (Breakpoint) This command allows the user to set a breakpoint anywhere in memory at nnnn. For example B 45A5 causes the break point to be set at 45A5 hex. Note that only one break point is allowed at the one time. If a breakpoint has already been set an error message will be displayed. A breakpoint is automatically cleared when it is reached. Because DBUG uses a single byte breakpoint (RST 30) programs will not crash inexplicably as happens with monitors which use 3 byte breakpoints. (also see the K command). |
| b | (Return to BASIC) Entering a [SHIFT] B will cause DBUG to pass control to BASIC. (Note that disk users should not issue |

this command UNLESS DBUG was invoked by DISK-BASIC).

C — (do Call) The "C" command causes DBUG to execute completely a CALL or RST (restart) and then return.

. — (Disassemble) This command has several uses, first it can be used to disassemble the area pointed to by the PC register, one instruction at a time. Secondly it is used to SKIP an instruction while single stepping through a program. The full-stop (period) key is used because it is conveniently located next to the ENTER key on the keypad. It will make disassembling by instruction much easier.

d — (Return to DOS) Entering a [SHIFT] D will cause DBUG to pass control to DOS. (Note that this command should only be used by disk users).

E nnnn — (Edit memory) The edit command is used to alter memory location nnnn. Each memory address starting at nnnn is shown followed by its current content displayed either in HEX or ASCII format. The E command has its own subset of control keys which is listed below. Note that the edit function will behave differently in ASCII mode (see A command above) compared to HEX mode. Note the following differences: The memory contents are displayed as a single alpha-numeric character and keyboard input is only a single character, again alphanumeric. This is useful if it is necesary to type text straight into memory.

EDIT SUB COMMANDS:

| | |
|---|---|
| UP ARROW | (ESC) Use this key to advance the edit pointer by one. |
| DOWN ARROW | (CTRL) Use this key to decrement to the previous memory location. |
| BREAK | Use BREAK to escape the EDIT command. |
| NOTE: | The arrow keys will have to be struck twice or followed by ENTER while in the HEX mode, as all input is in two digits while editing in HEX mode. |

G — (Goto) This command causes execution to resume at the current PC address. This is useful after a break point has been set.

UP-ARROW — (ESC) The up arrow key causes the PC register to be incremented by one. This is often useful.

DOWN-ARROW — (CTRL) The down arrow key causes the PC to be decremented by one.

J nnnn — (Jump) The jump command is used to jump to memory location nnnn, e.g. J 87FE will cause program execution to resume at 87FE hex.

K — (Kill breakpoint) As the name implies this command can be used to erase an unused breakpoint. This command must be

used to erase a breakpoint if it is desired to enter a new breakpoint while an old breakpoint is still in force.

L xxxxxx — (Load tape) This command loads a "system" format tape with a filename of up to 6 characters in memory. For instance L SAMPLE would search the tape for a program called SAMPLE and then load it. After loading is completed the entry point will be contained in the memory pointer at the lower left of the DBUG screen display. The "J nnnn" command can then be used to Jump to the program just loaded or the PC register can be set to the entry point value for single stepping or disassembly. MODEL 3 users may set the baud rate switch at 4211H to zero for 500 baud or to anything else for 1500 baud. This can be done using the Edit command.

M nnnn — (Memory) The M command will cause the memory pointer to be updated to point to nnnn.

P — (Print) This will cause the work display to be printed on the line printer.

R rp nnnn — (Register modify) This command allows the user to modify any register pair. Use the register pair name for rp, e.g. AF for register pair AF, SP for the stack pointer etc. The only non standard code is for the program counter which uses code : PR. The following codes are recognised: AF, BC, DE, HL, SP, IX, IY, PR. For example R PR 1234 will set the PC to 1234 hex.

S (clear Screen) Entering "S" will clear the screen and reset the cursor to the top of the screen.

[CLEAR] or / (Step) This is the single step command. The CLEAR key is used as it is located next to the ENTER key on the main key board. It causes the instruction pointed to by PC to be executed. (SYSTEM-80 users can use the / key instead)

W nnnn nnnn nnnn xxxxxx

(Write system tape) The W command writes a block of memory to tape, which can be reloaded with the L command (or SYSTEM command from BASIC). The three sets of hexadecimal numbers "nnnn" are in START END ENTRY format, "xxxxxx" means that a filename of up to 6 characters can be used. For example if we have a machine language program starting at 7000H, ending at 7100H with an entry point of 700FH in memory, and we wish to call it SAMPLE, then the correct statement to write it to tape would be:

W 7000 7100 700F SAMPLE

Note that the spaces separating the hexadecimal numbers and filename are required with this command.

; The ";" key is used to advance the memory pointer by 8 so that the next 8 bytes will be displayed.

| | |
|---|---|
| - | The "-" key is used to decrement the memory pointer by 8 so that the previous 8 bytes are displayed. |

## **** QUICK REFERENCE TABLE ****

| COMMAND | DESCRIPTION |
|---|---|
| A | ASCII mode. |
| B nnnn | Set break point at nnnn. |
| SHIFT B | Return to BASIC. |
| C | Execute Call or RST. |
| . | Disassemble (Bypasses instruction without executing it) |
| SHIFT D | Return to DOS ready mode. |
| E nnnn | Edit memory starting at nnnn. |
| G | Goto PC address. |
| J nnnn | Jump to nnnn. |
| K | Kill break points. |
| L xxxxxx | Load tape with file name xxxxxx in memory. |
| M nnnn | Set Memory pointer to nnnn. |
| P | Print work display on line printer. |
| R rp nnnn | Load register pair rp with nnnn. |
| S | Clears the screen. |
| CLEAR or / | Single Step. |
| W nnnn nnnn nnnn xxxxxx | Write memory to tape. |
| X | Hex mode. |
| ; | Advance memory pointer by 8. |
| - | Decrement memory pointer by 8. |
| DOWN ARROW | (CTRL) Decrement PC by 1. |
| UP ARROW | (ESC) Increment PC by 1. |

** EDIT SUBCOMMANDS **

| | |
|---|---|
| DOWN ARROW | (CTRL) Decrement Edit address by 1. |
| UP ARROW | (ESC) Increment Edit address by 1. |
| BREAK | Escape edit command. |

## ** ERRORS **

The message "ERROR" is displayed in the following cases:

1). The C command is used while the PC is not pointing to a RST or CALL.
2). An undefined command is issued.
3). Syntax error.
4). Trying to set a breakpoint while another is still in force.
5). When trying to set a breakpoint or single step through ROM.

The message "Illegal opcode" will be displayed if the current opcode to which the PC is pointing is an undefined code for the Z80.

## **** LOADING INSTRUCTIONS ****

The tape contains four versions of the DBUG program, each has a different memory location, they are:

| NAME | START | END | ENTRY | MEMORY-SIZE |
|---|---|---|---|---|
| DBUG6 | 5F30 | 6FF7 | 5F30 | 24290 |
| DBUG7 | 6F30 | 7FF7 | 6F30 | 28386 |
| DBUGB | AF30 | BFF7 | AF30 | 44770 |
| DBUGF | EF30 | FFF7 | EF30 | 61154 |

There is a copy of each version in the order shown on each side of the tape. The memory size shown is lower than the start value to leave some stack space for DBUG to use. Load DBUG using the SYSTEM command. As soon as loading is completed type a "/" to initialize and enter DBUG. After doing this two things will have happened. First, the SYSTEM command has been altered to cause entry into DBUG, second DBUG is entered. You can return to BASIC by typing [SHIFT] B or use DBUG immediately. From now on if the command SYSTEM is entered DBUG will appear. If it is desired to use SYSTEM for its normal function (to load a tape) use the DBUG tape facilities instead.

WARNING --- *** IF DBUG IS ALREADY RESIDENT DO NOT RELOAD OR TRY TO LOAD A DIFFERENT VERSION OF DBUG UNLESS THE SYSTEM IS SWITCHED OFF FIRST!!! OTHERWISE THE SYSTEM COULD CRASH ***

Disk users can load the programs from tape and create CMD files, these can than be directly executed from DOS. However if it is desired to use DBUG with DISK-BASIC, execute BASIC and set the memory size for the particular version of DBUG being used. Then, (assuming DBUG is on disk, with filename "DBUG") use CMD "DBUG" (NEWDOS) or CMD"I","DBUG" (TRSDOS), this

will cause DBUG to execute. DBUG can now be used as usual or aborted by using [SHIFT] B, which will re-enter the BASIC READY mode. To re-enter DBUG all that is required is to use the SYSTEM command from BASIC.

WARNING --- **** DO NOT USE THE "CMD" COMMAND TO RELOAD DBUG FORM DISK IF IT IS ALREADY RESIDENT AS THIS CAN CAUSE THE SYSTEM TO CRASH !!! USE THE "SYSTEM" COMMAND INSTEAD ****

## *** CONCLUSION ***

It must be recognised that trying to single step through DBUG itself can cause the computer to lock up. This will happen as soon as a break point is set by the single step processor into the break point handling routine itself, causing an endless loop. The TRSDOS DEBUG uses the same RST vector for setting breakpoints as DBUG does, this means that the two CANNOT be active at the same time. However the b and d commands restore the RST vector and DEBUG can then be used, but make sure that DBUG does not leave a breakpoint by using the K command before returning to BASIC or DOS. The b,d,G and J commands will leave a break point in force. If this is undesirable Kill them before invoking these commands. DBUG disables the interrupts, this means that the real time clock will not function. Note that all numbers in the symbolic representation of the instruction at the PC are in hexadecimal, including index numbers. For instance : LD HL,4590 means ; load HL with 4590 hex., and LD (IY+30),B means load the address pointed to by IY plus 30 HEX, with the contents of the B register. It must also be noted that the PC register is central to the operation of DBUG, DBUG will disassemble and single step using the value in the PC register. This means that the PC value will need to be altered by the user when required.

## LEVEL 2 ROM

# ASSEMBLY LANGUAGE TOOLKIT

*by Edwin Paay*

## FOR TRS-80 MODEL 1, MODEL 3 AND PMC-80/VIDEO GENIE/SYSTEM 80

Those who write Assembly Language programs for the TRS-80 are aware there is 12K of ROM full of useful machine language routines which, if they only knew how to get at them, would save reinventing the wheel time and time again. A number of books have been published which effectively give disassembled listings of the Model 1 ROM with a greater or lesser number of comments. They are helpful but leave the reader with a great deal of manual work to do before he or she can make use of ROM routines.

The Assembly Language Toolkit overcomes all that. It consists of two major components:- a ROM Reference Manual and DBUG, a debugging monitor program. In the ROM Reference Manual, the useful and usable routines in the Model 1 **AND** Model 3 ROMs are catalogued and are organised into subject specific tables so that you can quickly identify all the routines to carry out a given function and then choose the one which meets your requirements. Detailed information is supplied about each routine, including the effects of each on the various Z80 registers and how to use them in your own programs. The contents of system RAM are also detailed and you are shown how to intercept BASIC routines. With this knowledge you can add your own commands to BASIC, for instance or position BASIC programs in high memory — the only restriction is your own imagination!

The text is liberally illustrated with sample programs which show you how you can use ROM routines to speed up your machine-language programs and reduce the amount of code you need to write.

DBUG is a machine language program distributed on cassette but which also performs from disk. Designed to help speed up your assembly language program development, DBUG allows you to single-step through your program; has a disassembler which disassembles the next instruction before executing it or allows you to by-pass execution and pass on through the program, disassembling as you go; displays/edits memory in Hex or ASCII; allows Register editing; has the ability to read and write System tapes; and all this on the bottom three lines of your screen, thus freeing the rest of the screen for program displays. Four versions of DBUG are included in the package to cope with different memory sizes.