LEVEL II ROM REFERENCE MANUAL

By Edwin Paay

MICRO-8O PRODUCTS

WHO SHOULD BUY THIS MANUAL?

This manual is primarily intended for the Level II user who is interested in writing machine language programs, whatever his level of experience. Used in conjunction with reference texts on z 80 assembly language programming, it will enable the novice machine language programmer to quickly and simply make his Level II machine perform useful functions in machine language. On the other hand, the experienced machine language programmer who uses this manual 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 computer's memory space and will help him write faster, more compact programs in BASIC.

In short, if you use a Level II computer, then you need the LEVEL II ROM REFERENCE MANUAL.

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

COPYRIGHT (C) 1980 Edwin R Paay and MICRO-80 PRODUCT
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 Level II BASIC interpreter, which is the subject of this manual, are held by MICROSOFT. The author and publisher of this manual are in no way attempting to infringe these rights. The purpose of the LEVEL II ROM REFERENCE MANUAL is to assist users of LEVEL II microcomputers in using and understanding their machines.

*** 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 LEVEL II ROM REFERENCE MANUAL is sold in an "as is" condition and is not represented as being free from errors.

PRODUCED IN AUSTRALIA
by

MICRO-80 PRODUCTS

P.O. BOX 213 GOODWOOD SA 5034

PRINTED BY THE SHOVEL & BULL PRINTERS,' 312A UNLEY ROAD, HYDE PARK, SOUTH AUSTRALIA. 5061

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

Continued

***** TABLE OF CONTENTS (cont.) *****

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

***** INTRODUCTION *****

The BASIC interpreter in the Level II microcomputer is a machine language program which resides in a Read Only Memory (ROM) from address 0000 Hex to address 2FFFHex. The detailed contents of this ROM have never been officially revealed. True, some of the manuals explain how a few of the routines in ROM can be used in your own machine language programs but, until now, the vast bulk of this very clever 12K interpreter has been a mystery.

This manual reveals all. All the useful and useable ROM routines are explained. 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 II ROM REFERENCE MANUAL is in two parts. Part 1 contains an extended memory map which lists all useable routines and shows where the various BASIC command routines are in ROM. This serves as a useful reference to specific addresses in ROM and also as an adjunct to a disassembled listing.

Part 2 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 2 also contains discussion on the ACC, AACC and 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 CPU is always referred to as A register and never as accumulator. This is done to avoid confusing it with the area in memory referred to as ACCumulator).

In preparing this manual, we have assumed that our readers understand the Z-80 instruction set and have some experience of programming with the aid of an editor/assembler. This manual is in no way a substitute for a good text on Z-80 assembly language programming, rather, it will help the Level II user to obtain the most from such texts.

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

MEMORY LOCATION	COMMENTS
0000-0002	Disables the interrupts, clears the A register, then jumps to initialisation routine at 674H.
0008	(RST 8H) Jumps to 4000H. 4000H passes control to 1C96H. This routine is used for scanning strings. 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 is used to look for expected characters in a string and then return with (HL) pointing to the next non-blank character. (see RST l0H) (BC and DE registers unaffected.). This routine can be used by CALLing 1C96H or RST 8H.
0010	(RST l0H) 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 CHAR'S 9 and 10 (shifted left and down arrows 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 (ASCII 30H) or 3AH (":"). (BC and DE registers are unaffected) This routine can be used by CALLing 1D78H or RST l0H.

0018 (RST 18H) Jumps to lC90H through 4006H. This routine can be called by using RST 18H or CALL lC90H. 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 CP). All registers are unchanged except for A.

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 will be 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 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 00H 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.

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 pair is used by the routine.

0038 (RST 38H) This location will pass control to 4012H. This location is only used by a Disk system.

003B Character LPRINT routine. Same as 33H but outputs to line printer. (Contents of A register will be printed.)

0049 Character input routine. This routine is the same as

2BH except that it will not return until a key is pressed, which often makes it often more useful than 2BH. Character is returned in the A register (AF and DE used).

0050-005F This is a table of control characters used by BASIC.

50	Carriage Return (ENTER)
52	CLEAR
54	BREAK
56	Up Arrow
58	Down Arrow
5A	Left Arrow
SC	Right Arrow
SE	SPACE

0060 This is a delay loop. 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).

0069-0074 This part of the initialization routine checks to see if a disk drive is connected. If so, it will jump to 00H. (This is why the reset button will reinitialize DOS.)

0075 This is part of the Level II initialization procedure. It moves a block of memory from 18F7H to l9lEH up to 4080H to 40A7H. (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.)

0091-0104 The rest of the initialization routine. Asks MEMORY SIZE ?, sets the memory pointers accordingly and prints RADIO SHACK LEVEL II BASIC , then it jumps to lAl9H which is the entry point for the BASIC command mode.

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, 0135, 0138 These are the entry points for the POINT, SET and the RESET commands in that order, see Part 2 for more data on the graphics routines.

0150 This is a suitable entry point for the graphics routines. (see Part 2)

01C9 A CALL lC9H 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.

01D9, 01E3 and 01ED Output a pulse to the cassette recorder.

01DF, 01E9 and 01F3 Delay loop between pulses.

01F8 Turns cassette recorder off.

0212 CALL 212H will define which cassette is to be used. Put 00H in A register to turn On cassette 1, or O1H to turn on cassette 2. (BC,DE and HL are unchanged)

022C Blinks asterisk in top right corner. This can be used as a subroutine. AF register pair is used.

0235 This routine will read 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.

0241 Routine waits for timing pulse, and then performs a timing loop. When the time is up it tests the tape for a bit, which will be "1" if present and "0" if not. A CALL 241H is used by 235H eight times to input one byte.

0264 Writes the byte in the A register to tape. BC, DE and HL are unchanged by a CALL 264H.

0287 Writes tape leader and the A5H sync byte. DE and HL are unchanged.

0296 Reads from tape until the leader is found, then keeps going until it is bypassed and the sync byte (ASH) is found, when it returns. DE, BC and HL are unchanged by this.

029F Places the double asterisk in the right top corner to show that the sync byte has been found.

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.

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 doesn't destroy the contents of the DE register pair. This means that all the general purpose registers are saved, which is often 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 (F0H), 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 and B contains the length of the string. (RST 10H can be used to make HL point to the first character of the string, if required.)

039C This is the LPRINT routine. All registers are saved. The byte to be printed should be in the A register.

03E3 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 This is the video driver. The character to be displayed should be in the C register. This routine handles scrolling etc.

04C3 Changes display to 64 character mode (A register is used).

04F6 Changes display to 32 character mode. A and HL registers used.

057C 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 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 (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.

06CC This is an alternative re-entry point into BASIC. A JP 6CCH is often better than a jump to lA19H as the latter sometimes does strange things to any resident BASIC program.

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 2 of this manual for information on the ACC). This part of the program loads the BCDE registers with the 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.

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 2.)

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=-l, 0 or 1 depending on sign and value of ACC be fore 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 > 0 or A=FFH if A < 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 register pairs. (BCDE=ACC). A is unchanged.

09C2 This routine will load 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)).

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>BCDE or A=FFH if ACC<BCDE.

0A39 Integer compare. Compares HL with DE. After execution, A=0 if HL=DE, A=1 if HL>DE or A=FFH if HL<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 > AACC or A=FFH if ACC <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 that 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.

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.

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.

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. The result is returned in both the ACC and the HL register pair.

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).

0DAl Double-precision multiplication (ACC=ACC*AACC).

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 E65H above, except that it fixes the ACC and NTF to the smallest possible number type.

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 values only should be used. (ACC=SQR (ACC)).

1439 EXP routine. Single-precision only. (ACC=EXP (ACC)).

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)).

l5BD 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 into BASIC command mode entry point. (see 6CCH also).

lAF8 This routine fixes the line pointers in a BASIC program. This is useful, for instance for a renumber program which has to move BASIC program lines from one location in memory to an other, which means that the line pointers would no longer be valid. This routine will fix them. Registers A, HL and DE are used.

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 points to the start of the required line, and the HL register points to the start of the next line. HL, AF and BC are used.

1B49 Entry point of the NEW command.

1BB3 This is the last of the general purpose input routines. This routine functions identically to the 361H routine with the exception that it prints a "?" on the screen (like INPUT does with BASIC) before allowing input from the keyboard.

1C90 The RST 18H code is located here. (Unsigned compare (HL-DE))

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 1OH 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.

1E06 DEFSNG entry point.

1E09 DEFDBL entry point.

1E3D This routine tests the value pointed to by the HL register pair and sets the C flag if it is an ASCII letter value. Otherwise it resets the C flag.

1E4A FC ERROR entry point.

1E5A Converts numeric ASCII string pointed to by the HL register pair, to HEX 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 equals 00 or 3A. Z is reset if any other delimiter is used. If there is no string at the location pointed to by HL the routine will return a MO ERROR (missing operand). If the result in the DE register pair exceeds FFFFH 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.

1F05 DATA entry point.

1FF4 ERROR entry point.

2003 UE ERROR entry point.

2039 IF entry point.

2067 LPRINT logic.

206F PRINT logic.

2376 PRINT @ logic.

2137 TAB logic.

2178 ?REDO message string.

219A INPUT logic.

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 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 formatted for RUN mode in order to use this routine. (See sample program in Appendix 1 for an application of this routine).

2490 Integer divide. (ACC=DE / HL) 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 first three bytes which contain the string length and 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 20H code is located here.

25F7 OR logic.

25FD AND logic.

2608 DIM logic.

260D This is the variable location and creation logic. This routine will return the address of a variable in memory or create 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.

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 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 (3AH) then the Z flag will be set. Any other delimiter will cause the Z flag to be reset.

2B29 LLIST logic.

2B2E LIST logic.

2BC6 DELETE logic.

2BF5 CSAVE logic.

2ClF CLOAD logic.

2CAA PEEK logic.

2CA5 "BCD" message string.

2CB1 POKE logic.

2CBD USING logic.

2E60 EDIT logic.

2F0A This routine prints a string of text to the display, printer or tape. it uses 32AH to do this (see 32AH routine for further rules), HL must point to the first character of the string. (409CH must be set before calling this routine, see 32AH). String must be delimited with a zero byte.

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

**** RESERVED RAM AND DEVICE ADDRESSES ****

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.

3801-3880 Keyboard area. (see special section on keyboard for more information).

3C00-3FFF Video display memory.

4000 Jump vector for RST 8H.

4001 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. (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 key board scanning 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) .

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.

4029 Current line number being printed.

4033 A jump to a machine language routine can be placed here. (see section on program intercept and 4015H)

4036-403C Small buffer for keyboard decoding routine (used for keyboard rollover).

403D Print size flag. (0=64 characters F8=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.

4093 INPut routine.

4094 Port number.

4096 OUT routine.

4097 Port number.

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 Start of 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.
40AB	NSB (Next most Significant Byte) of seed. (see lD3H)
40AC	MSB of seed.
40AF	NTF this is the Number Type Flag. This address tells BASIC what type of number is contained in the ACC. (2=integer, 3=string, 4=single and 8=double-precision.)
40B1	Top of BASIC memory pointer.
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.

40EA Used by RESUME.

40EC EDIT line number.

40EE Used by RESUME.

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.

4101-411B Variable type declaration table. 2=INT 3=String, 4=single, 8=Doub1e.

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.

DOS COMMAND ENTRY POINT TABLE

(for actual entry points, see map in Part 2 of this manual)

4152	CVI
4155	FN
4158	CVS
415B	DEF
415E	CVD
4161	EOF
4164	LOC
4167	LOF
416A	MKI$
416D	MKS$
4170	MKD$
4173	CMD
4176	TIME$
4179	OPEN
417C	FIELD
417F	GET

4182	PUT
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.
4lBB	Intercept for program initialization.
41E8-42E8	I/O Buffer area.
42E9	Level II BASIC programs start here.
6A24	Disk BASIC programs start here.

***** PART 2 *****

In this part of the manual, the ROM routines which were described in numeric order in Part 1 are organized by function in a series of tables. This should enable the user to quickly choose the most suitable routine for his purpose.

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

- if a register is not mentioned specifically, then the routine destroys its content

- the addresses given in the Tables are the CALL addresses

Note: There is no Table 8 in this Part. Table 8 was amalgamated with Table 7 during typesetting.

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

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 arithmetic 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 in to 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
ORGANISATION 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. 128 is added to the exponent. 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 as 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.				
70BH	ACC = (HL) + ACC	ACC, (HL)	ACC	NTF = 4,Single
710H	ACC = (HL) - ACC	ACC, (HL)	ACC	NTF = 4,Single
713H	ACC = BCDE - ACC	ACC, BCDE	ACC	NTF = 4,Single
716H	ACC = BCDE + ACC	ACC, BCDE	ACC	NTF = 4,Single
847H	ACC = BCDE * ACC	ACC, BCDE	ACC	NTF = 4,Single
8A2H	ACC = BCDE / ACC	ACC, BCDE	ACC	NTF = 4,Single
INTEGER.				
BD2H	ACC = DE + HL	DE, HL	ACC, HL	NTF = 2, int
BC7H	ACC = DE - HL	DE, HL	ACC, HL	NTF = 2, int
BF2H	ACC = DE * HL	DE, HL	ACC, HL	NTF = 2, int
2490H	ACC = DE / HL	DE, HL	ACC, HL	NTF = 4,Single
DOUBLE PRECISION.				
C7OH	ACC = ACC - AACC	ACC, AACC	ACC	NTF = 8, dbl
C77H	ACC = ACC + AACC	ACC, AACC	ACC	NTF = 8, dbl
DA1H	ACC = ACC * AACC	ACC, AACC	ACC	NTF = 8, dbl
DE5H	ACC = ACC / AACC	ACC, AACC	ACC	NTF = 8, dbl

Note : all reg's are used.

TABLE 3.

ARITHMETIC FUNCTIONS AND NUMBER CONVERSIONS.

ADDRESS		FUNCTION	INPUT	OUTPUT
809H		LOG (ACC)	ACC, NTF=4	ACC, NTF = 4
977H	&	ABS (ACC)	ACC, NTF=2,4 or 8	ACC, NTF=2,4 or 8
982H	$#	- (ACC)	ACC, NTF=4 or 8	ACC (Negate)
98AH		SGN (ACC)	ACC, NTF=2,4 or 8	ACC= -1,O,l,NTF=2
A7FH		CINT (ACC)	ACC, NTF=2,4 or 8	ACC and HL, NTF=2
AB1H		CSNG (ACC)	ACC, NTF=2,4 or 8	ACC, NTF=4
ADBH		CDBL (ACC)	ACC, NTF=2,4 or 8	ACC, NTF=8
B26H	*	FIX (ACC)	ACC, NTF=2,4 or 8	ACC, NTF=2 or 4
B37H	*	INT (ACC)	ACC, NTF=2,4 or 8	ACC, NTF=2 or 4
C5lH	#	- (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 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 reg.'s 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 labeled "ADDRESS" is the call address, the column labeled "FROM" is the source (usually a register pair is used to point to a location in memory) and the column labeled "TO" is the destination. The column labeled "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 labeled "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
9A4H	*	ACC	(SP)	4	A,BC,HL saved
9B1H		(HL)	ACC	4	HL = HL + 4
9B4H		BCDE	ACC	N/A	BC,HL saved
9BFH		ACC	BCDE	N/A	N/A
9C2H		(HL)	BCDE	4	HL = HL + 4
9CBH		ACC	(HL)	4	HL = HL + 4
9CEH		(DE)	(HL)	4	HL = HL + 4 DE = DE + 4
9D2H		(HL)	(DE)	NTF	HL = HL + NTF DE = DE + NTF
9D3H		(DE)	(HL)	NTF	HL = HL + NTF DE = DE + NTF
9D6H		(DE)	(HL)	A A =< 256	HL = HL + A DE = DE + A
9D7H		(DE)	(HL)	B B =< 256	HL = HL + B DE = DE + B
9F4H		AACC	ACC	NTF	N/A
9FCH		ACC	AACC	NTF	N/A
A9AH		HL	ACC	N/A	N/A

* 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 req. pair.

COMPARE & 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. The flags are set or reset however, 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 req. 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 ISN'T
1D7D    FE20        CP      20H         ;       ;IS CHAR ASCII SPACE ?
1D7F    CA78lD      JP      Z,1D78H     ;       ;LOOP TQ BYPASS SPACE
1D82    FE0B        CP      OBH         ;       ;IS IT > OAH ?
1D84    3005        JR      NC,lD8BH            ; 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.
```

TABLE 5.
COMPARE & TEST ROUTINES.

NO	ADDRESS	FUNCTION	COMMENTS ROUTINE TYPE (NTF)
1	0994H	SGN (ACC)	* NTF =2,4 or 8, ERROR if NTF = 3 ALL REG'S SAVED
2	0A0CH	ACC-BCDE	* NTF = 4 ONLY ALL REG'S SAVED
3	0A39H	HL-DE	* TWO'S COMPLEMENT ALL REG'S SAVED
4	0A4FH	ACC-AACC	* NTF = 8
5	0A78H	AACC-ACC	* NTF = 8
6	1C90H	HL-DE	BC, DE AND HL SAVED UNSIGNED COMPARE
7	1C96H RST 8H	(HL)-((SP))	BC,DE SAVED
8	1D78H RST 10H	(HL)	BC, DE SAVED C and Z flags used ,HL incremented see text.
9	25D9H RST 20H	NTF	A = NTF – 3,Z & S flags valid C set if NTF <> 8, C reset if NTF = 8
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 <> 3, TM ERROR BC,DE and HL saved
This routine examines the byte pointed to by HL for character type			
12	lE3DH	(HL)	C flag set if (HL) =ASCII letter value or reset otherwise.

* A reg. = 0 if equal, A reg. = 1 if > 0, A reg. = FFH if < 0

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

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 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 l0H 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 l0H instructions.

* * * * * * * * *

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.

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. (00H 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 by 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 HEXADECIMAL value. The result is placed in the DE register pair. For more information see Part 1.
2B1CH	(HL)	A	The main function of this routine is to convert a numerical ASCII string pointed to by the HL register pair to HEXADECIMAL. The result is placed in the A register. For more information see Part 1.

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 lBB3H routines pass control to 4lAFH 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 for the key which is pressed, in the A register. A will return with zero (0) if no key was pressed during the scan. BC and HL are saved.
0049H	reg. A	This routine is the same as 2BH except that it will not return until a key is pressed. 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 DE req. pair.

STRING INPUT ROUTINES.

NOTE THAT ALL THE ROUTINES IN THIS TABLE WILL DISPLAY THE DATA TO THE SCREEN AS IT IS TYPED IN.

ADDRESS	COMMENTS
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 B req. will contain the string length and the HL req. 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.

STRING OUTPUT ROUTINES

There are several output routines available in ROM. Some are general purpose output 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.

TABLE 9.

SINGLE BYTE OUTPUT ROUTINES.

ADDRESS	COMMENTS
0033H	This routine will output a byte from the A register to the video display. DE is destroyed.
OO3BH	This routine is the same as 33H except that output goes to the line printer. DE is destroyed.
032AH	* This routine will output a byte from the A register to the display, printer or tape depending on the state of 409CH. (see text).
033AH	This routine performs the same function as 33H, the only difference being that the DE register pair is saved. This means that all the general purpose registers are saved.
039CH	This routine outputs a byte from the A register to the line printer, it is the same as 3BH except that the DE register pair is also 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
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 performs the same function and has the same requirements as the 28A7H routine. The only difference is that the output string can only be terminated with a zero byte and will not accept a quote like 28A7H does for a delimiter.

- 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 by a comma then they will be multiplied and the result printed on the screen.

```
10          ORG     7000H
20 START    CALL    1BB3H      ;INPUT X,Y
30          RST     10H        ;FIND FIRST CHAR
40          CALL    0E6CH      ;PUT X VALUE IN ACC
50          PUSH    HL
60          CALL    0AB1H      ;CSNG
70          CALL    9BFH       ;STORE TO BCDE REG.S
80          EXX                ;SAVE REG.S
90          POP     HL
100         RST      8H        ;CHECK FOR COMMA
110         DEFB    ','
120         CALL    0E6CH      ;PUT Y VALUE IN ACC
130         CALL    0ABlH      ;CSNG
140         EXX                ;RESTORE REG.S
150         CALL     847H      ;MULTIPLY X AND Y
160         LD      HL,BUFR
170         CALL    0FBDH      ;PUT RESULT IN BUFFER
180         XOR     A          ;SET FLAG FOR
190         LD      (409CH),A  ;VIDEO OUTPUT
200         CALL     28A7H     ;PRINT RESULT
210         LD      A,0DH
220         CALL     33H       ;PRINT CARRIAGE RET.
230         JR      START      ;DO IT AGAIN
240 BUFR    DEFS    240        ;240 BYTE BUFFER HERE
250         END     START
```

TAPE I/O ROUTINES

Using tape I/O in a machine language program is straight forward. All tape I/O can be handled by a series of calls. Before using the tape I/O routines it might be a good idea to read the data format section to find out how data is actually stored on tape. Also note that the user will have to determine when to stop reading from tape. The data format section can be consulted, to show what the end of file pointers are.

* The general purpose output routines 32AH and 28A7H can also be used to output to tape, however 212H and 287H will have to be called to define drive and write leader as normal.

DEMO TAPE I/O ROUTINE

This is a tape I/O routine which can either read or write 255 bytes. 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
40          LD      HL,5000H    ; START ADDRESS
50          LD      B,0FFH      ; WRITE 255 BYTES
60 LOOP     LD      A, (HL)     ; GET BYTE FROM MEM
70          CALL    264H        ; WRITE IT TO TAPE
80          DJNZ    LOOP        ; LOOP TILL DONE
90          CALL    1F8H        ; TURN CASSETTE OFF
100         JP      6CCH        ; RETURN TO BASIC
110         END     START
```

The read routine will be identical except for line 70 which will read:

```
70          CALL    235H        ; READ A BYTE FROM TAPE
```

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 0lH to turn recorder 2 on. BC,DE & HL are saved
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 asterisks 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.
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.
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 doesn't 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. delimited with :,) or zero byte.	ACC, NTF of ASCII string
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 ORGANISATION

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

NTF This is the number type. (also length of data)

2nd char of variable name.

1st char 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 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 (ie. the number of bytes used). The third byte contains the number of dimensions used, next there are 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 become necessary at times to test for errors. Finding errors is not a problem, but letting the outside world know what sort of error has occurred can be tedious and cost both program space and space to store the error message strings. To overcome this, BASIC's 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 handed back to the BASIC command mode. Which brings us to another related subject; whenever an error occurs while in the BASIC mode, control is passed 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 which will print the error messages in full instead of the abbreviated form that Level II uses.

TABLE 13.

ERROR ROUTINE ENTRY POINTS.

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	(UNPRINTABLE ERROR)
27EDH	BS ERROR	(BAD SUBSCRIPT)
2831H	ID ERROR	(ILLEGAL DIRECT)
2BA1H	ST ERROR	(STRING FORMULA TO 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 cursor location contained by 4020H to 4021H and CALL 57CH.

TABLE 14.

VIDEO CONTROL ROUTINES.

ADDRESS	INPUT	COMMENTS
01C9H	N/A	Performs the CLS function
04C3H	N/A	This routine will change the display back to 64 char.'s per line.
04F6H	N/A	This routine will change the display to 32 char.'s per line
057CH	HL = cursor pos. or address from which erasing is to start.	Clear screen to end of frame routine.

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 called 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 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 TO STACK.
150              HALT                ;FINISHED.
160 GRAFIX       PUSH    HL          ;PUSH FLAG.
170              PUSH    BC          ;PUSH X COORDINATE.
180              LD      HL,188CH    ;POINT HL TO DUMMY STRING.
190              JP      150H        ;JUMP TO GRAPHICS LOGIC.
200              END
```

Now, let's 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. That was all straight forward, line 130 however needs explaining. This line loads the H register with 80H. 80H is the flag for the SET function. The graphics routine can perform the POINT, SET or RESET function depending on the flag byte passed to it in the H register Note the following flag values for the different functions:

80H = SET

0lH = RESET

00H = POINT (the POINT logic will return with 0 in the ACC if block is SET or with FFFFH if block is RESET)

Line 140 then CALLs the code labeled GRAFIX. Stop and think about this for a while because there is more to it then first appears. A CALL is used here to place the return address on the stack for the RST 10H logic which is the very last routine called by the graphics routine in ROM. Whatever process is used to do this a return address MUST be pushed on the stack at this stage.

Line 150 is the location of the return address pushed by the CALL statement in line 140. The program will just stop here when its task is finished. Line 160 pushes the flag byte onto the stack and line 170 pushes the X coordinate. These two values must be pushed to the stack in the order shown because the graphics routine will POP them off the stack when it needs them. Line 180 loads the HL register pair with the address of a dummy string. This string is located in ROM at 188CH and is ")+". It is not necessary to use a string out of the ROM area but it saves space in our sample program. There are numerous strings in ROM suitable for this purpose. A dummy string is needed because when the RST 8H is called by the graphics routine, it looks for a right bracket. A dummy string with a bracket followed by any non blank character is therefore needed. The reason the RST 8H routine looks for a right bracket ")" is because the graphics routine is a part of the BASIC decoding logic and it is looking for the bracket associated with the BASIC statements SET, POINT and RESET. Finally line 190 will pass control to the graphics logic. After reading this section carefully graphics should pose no problem. Of course, if more than one block has to be set, then a routine such as this will have to be placed in some form of loop.

KEYBOARD MEMORY

The title of this section is actually not quite correct. The keyboard area from 3800H to 3BFFH is not memory at all but a matrix of buffered switches. If the format of this matrix is known, it is quite simple to use the keyboard for input directly instead of the input routines. Sometimes, it is necessary to scan the keyboard quickly for a particular key (such as the BREAK key) and it will be necessary to know where to look. The following table shows the matrix format.

BIT NUMBER	7 (80)	6 (40)	5 (20)	4 (10)	3 (08)	2 (04)	1 (02)	0 (01)
ADDRESS								
3801	G	F	E	D	C	B	A	@
3802	O	N	M	L	K	J	I	H
3804	W	V	U	T	S	R	Q	P
3808						Z	Y	X
3810	' 7	& 6	% 5	$ 4	# 3	" 2	! 1	0
3820	? 0/	> .	= -	< ,	+ ;	* :	) 9	(8
3840	SPC	RA	LA	DA	UA	BRK	CLS	ENT
3880								SHFT

(SPC = SPACE, BRK = BREAK, CLS = CLEAR, SHFT = SHIFT, ENT = ENTER, RA = RIGHT ARROW, LA = LEFT ARROW, UA = UP ARROW, DA = DOWN ARROW.)

Let us go through an example on how to use this Table. First let us assume we want to check if the BREAK key is down. Now find the row containing the BREAK key in the table, then note the address at the left of this row. This is the address from which to read. Then go up from the location of the BREAK key in the Table to see which bit will be set if the BREAK key is pressed. A program which checks for the BREAK key would look like this:

```
100   LD      A,3840H      ; READ FROM ROW CONTAINING BREAK
                            KEY.
110   AND     04H          ; MASK AND SET FLAGS
120   JP      NZ,BREAK     ; BREAK KEY IS DOWN
130   ……                   , CONTINUE AS BREAK KEY IS NOT
                            DOWN
```

Line 100 reads from the row indicated in the table for BREAK key. Line 110 will only reset the Z flag if the BREAK key is pressed down. (note that the value to AND with is given in the column containing the key in question.) Line 120 then will jump to a location called "BREAK" only if the BREAK key is down.

* *

DOS LINK ADDRESSES

The Level II 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 then possible with the Level II 12K interpreter: In order to allow for this the Disk BASIC link areas were created. Whenever BASIC finds a Disk BASIC command in a program (or typed directly from the keyboard) it will jump to a unique location in reserved RAM. In the case of Level II these locations contain the L3 ERROR entry points. Disk BASIC loads these locations with addresses of routines which can execute the particular commands. The table below gives these locations for both TRSDOS 2.2 and TRSDOS 2.3.

		ENTRY POINT	
LINK ADDRESS	COMMAND	TRSDOS 2.2	TRSDOS 2.3
4152H	CVI	5E46H	5Fl8H
4155H	FN	558EH	558EH
4158H	CVS	5E49H	5FlBH
415BH	DEF	5655H	5655H
415EH	CVD	SE4CH	5F1EH
4161H	EOF	6lEBH	62B8H
4164H	LOC	6231H	62FEH
4167H	LOF	6242H	630FH
416AH	MKI$	5E2DH	5EFFH
416DH	MKS$	5E30H	5F02H
4170H	MKD$	5E33H	5F05H
4173H	CMD	56C4H	56CBH
4176H	TIME$	5714H	5745H
4179H	OPEN	6349H	6434H
417CH	FIELD	60ABH	61AFH
417FH	GET	627CH	6355H
4182H	PUT	627BH	6354H
4185H	CLOSE	606FH	6173H
4188H	LOAD	5F7BH	606EH
418BH	MERGE	600BH	6109H
418EH	NAME	6346H	6544H
4191H	KILL	63C0H	6521H
4194H	&	58B7H	5913H
4197H	LSET	60E6H	620BH
419AH	RSET	60E5H	620AH
419DH	INSTR	582FH	588BH
4lA0H	SAVE	6044H	6148H
41A3H	LINE	5756H	5786H

Level II users can use these addresses to make the machine jump to a machine language program when BASIC finds a particular Disk BASIC command.

INTERCEPT ADDRESSES

There are a few 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 a very useful interface address as well, 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 line to be processed next. The keyboard scanning routine can be intercepted by loading location 4015H with zero and putting a jump to the entry point of your machine language routine. This method can be used to intercept characters before BASIC can respond to them. In order to do this however the user must CALL the keyboard driver routine at 3E3H first. To give an example assume that we wish to disable the BREAK key, the code that the BREAK key returns is 01H. A program to do this will look like this:

```
10          ORG   4015H
20          DEFB  0
30          ORG   4033H
40          DEFB  C3H       ; C3 = JUMP
50          DEFW  START     ; LOAD JUMP VECTOR
60          ORG   7000H
70 START    CALL  3E3H      ; GET CHARACTER
80          CP    0lH       ; IS IT BREAK KEY
90          RET   NZ        ; CARRY ON IF NOT BREAK
100         XOR   A         ; CLEAR A REG. TO IGNORE BREAK KEY
110         RET             ; CARRY ON
120         END
```

A call 3E3H will return with the key which is pressed in the A register. It is therefore a simple matter to Compare if the character returned is the one you are looking for.

MISCELLANEOUS

This section handles all the leftovers which don't fall under any particular group. The routines are handled according to their location in memory.

TABLE 15.

ADDRESS	COMMENTS.
0060H	The delay loop is located here. This routine uses the BC register pair as a loop counter and loops until the BC register is decremented to zero. The time delay is the value in BC multiplied by 14.65 microseconds. The A register is destroyed.
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 BASIC from a machine language routine
lAl9H	This is a re-entry point for BASIC also. (6CCH is however, recommended instead).
lAF8H	This is a very useful routine. 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.
lB2CH	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.

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 for tape only. 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.

BASIC PROGRAM FORMAT.

* Leader	consisting of 256 zeros.
* A5	Sync byte.
* D3 D3 D3	Basic header.
* Name	File name (one character long)
LSB	Line pointer. Points to location of start
MSB	of next line.
LSB	Line number in hexadecimal.
MSB	
•	Actual program line of varying length is
•	stored here.
00	Zero byte signifies end of line.

New line starts here (starting with line pointer) OR

00 00	End of BASIC program. This is marked by two zero bytes after the end of line marker.

SYSTEM TAPE FORMAT.

* Leader	
* A5	Sync byte
* Name	Six byte long ASCII file name. If name is less than six bytes long it will be padded with blanks.
3C	Block header code.
XX	Block length from 1 to 256 bytes. (0 = 256)
LSB	Starting location of block in memory.
MSB	
•	Block of data stored here. (varying length)
XX	Checksum. The total of starting address and all data in this block with any carry ignored.

The block from 3CH onward is repeated until all the program is complete.

78	End code.
LSB	Entry point for program.
MSB	

FORMAT OF A SOURCE FILE FROM EDTASM.

* Leader	
* A5	Sync byte.
* D3	Start code.
* Name	Six byte long name. Names shorter than six bytes are padded with blanks.
Line number	The line number will be stored in ASCII and is five bytes long. (note that this is not real ASCII code because the eighth bit is set)
20	Blank spacer.
•	ASCII coded source code stored here.
0D	Carriage return marks end of line.
1A	End of file marker.

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

Leader	
A5	Sync byte.
XX	Sign of the data. Will be 20H if positive or 2DH if negative.
	ASCII coded data stored here.
20	End Of Field Code.
2C	Field separator. (ASCII for comma)

More data may be placed here

0D	Carriage return marks end of data.

ADDRESSES USED BY EDTASM.

At this stage it might we 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 to tape.

ADDRESS	USE
4113	End of memory pointer.
4115	Start of memory pointer.
41C3	Start of symbol table pointer.
4301	Keyboard driver address pointer.
45AA	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 (4288H) used by BASIC as the buffer. 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 except that the Stack Pointer must contain the same value 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, (4230H onwards) and the BASIC program pointers set past it. If a program is placed in low memory the start of BASIC pointer 40A4H and the pointers 40F9H, 4OFCH and 4OFDH will have to be set a couple of bytes past the end of your machine language routine. 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 which make no ROM calls or with machine language subroutines used with BASIC through the USR function. The problem stems from the fact that DOS does not initialize the BASIC pointers and jump vectors, 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 will use the SYSTEM command to jump to the entry location of their program. This is one way of doing things but is somewhat tedious because the user has to remember the entry point address in decimal for the SYSTEM command. There is however, a different 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 BASIC initializing procedure when BASIC is called up. This means that the line printer driver address at 4026H and 4027H can be loaded with the entry point to a machine language program then after BASIC is initialized all that is required is to type 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 as soon as control is passed to the machine language program.

PORT 255

Port 255 has several different applications in the TRS-80. It is used for cassette recorder control and I/O. 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 Will cause a zero voltage to be written to tape. 01 Writes a positive voltage to tape. 10 Writes a negative voltage to tape.

BASIC program which sets each control bit in turn, demonstrating the functions of port 255.

```
10 FOR X = 1 TO 4: READ N: OUT 255, N

20 PRINT N: FOR D = 0 TO 500: NEXT D

30 NEXT X: END

40 DATA 8,4,2,1
```

APPENDIX 1
SAMPLE PROGRAM

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 two's complement form. Since the routine is going to be part of the BASIC interpreter, it is reasonable to put the routine 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 variables pointer and the free space pointer accordingly. Note that the start of BASIC pointer must point to two bytes before the end of BASIC pointer and that the byte before and two bytes after the start of BASIC memory must be zero to show BASIC that there is no program in memory. 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 same range as BASIC integer values).

Although the program is not terribly exciting, 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 commas) 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 the string address. Line 200 to 220 checks if the string length is zero and generates an ILLEGAL FUNCTION CALL if it is. Line 240 to 270 gets the string address and checks if the control character is a "B". The rest of the program is straightforward and 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.

The new BASIC command can be used in the following way:

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

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

```
10              ORG     433FH
20 INIT         LD      HL,FINISH+1
30              XOR     A               ;CLEAR A
40              LD      (HL),A          ;ZERO MEMORY LOCATIONS
50              INC     HL
60              LD      (HL),A
70              LD      (40A4H),HL      ;FIX THE START OF BASIC
80              INC     HL              ;MEMORY POINTER
90              LD      (HL),A
100             INC     HL
110             LD      (40F9H),HL      ;FIX VARIABLE AREA POINTER
120             LD      (40FBH),HL      ;FIX ARRAY POINTER
130             LD      (40FDH),HL      ;FIX FREE SPACE POINTER
140             LD      HL,ENTRY        ;GET ENTRY ADDRESS
150             LD      (4174H),HL      ;FIX CMD LINK VECTOR
160             JP      6CCH            ;RETURN TO BASIC
170 ENTRY       CALL    2337H           ;EVALUATE EXPRESSION
180             PUSH    HL
190             CALL    29D7H           ;GET STRING ADDRESS ETC
200             LD      A, (HL)         ;GET STRING LENGTH
210             OR      A               ;SET FLAGS
220             JP      Z,1E4AH         ;IF ZERO FC ERROR
230             INC     HL
240             LD      E,(HL)
250             EX      DE,HL           ;STRING ADDRESS IN HL
260             LD      A, (HL)         ;GET FIRST CHARACTER
270             CP      'B'             ;IS IT B  ?
280             JP      NZ,1E4AH        ;ONLY ACCEPT CMD"B"
290             RST     10H             ;FIND NEXT CHARACTER
300             RST     8H              ;IS IT A COMMA ?
310             DEFB    ','
320             CALL    2540H           ;PUT VALUE OF VAR IN ACC
330             CALL    0A7FH           ;CONVERT TO INTEGER
340             CALL    BINOUT          ;DISPLAY BINARY BYTE
350             LD      H,L
360             CALL    BINOUT          ;DISPLAY LSB BYTE
370             POP     HL
380             RET                     ;ALL DONE
390 BINOUT      LD      B,8             ;8 BITS IN BYTE
400 LOOP        XOR     A
410             SLA     H               ;SHIFT BIT INTO C FLAG
420             ADC     A,30H           ;ADD BIT AND CONVERT TO ASCII
430             CALL    33H             ;DISPLAY IT
440             DJNZ    LOOP            ;LOOP TILL 8 BITS DONE
450             LD      A,20H
460             CALL    33H             ;DISPLAY SPACE
470 FINISH      RET
480 END         INIT
```

APPENDIX 2

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, linefeed 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.

BINARY	HEX.	DEC.	COMMENTS
00000000	00	0	NULL
00000001	01	1	BREAK KEY
00000010	02	2	
00000011	03	3	
00000100	04	4	
00000101	05	5	
00000110	06	6	
00000111	07	7	
00001000	08	8	BACK SPACE (left arrow key)
00001001	09	9	TAB (right arrow key)
00001010	0A	10	LINE FEED (down arrow key)
00001011	0B	11	
00001100	0C	12	FORM FEED
00001101	0D	13	CARRIAGE RETURN (enter key)
00001110	0E	14	
00001111	0F	15	
00010000	10	16	
00010001	11	17	
00010010	12	18	
00010011	13	19	
00010100	14	20	
00010101	15	21	
00010110	16	22	

00010111	17	23	CHANGE TO 32 CHAR./LINE MODE
00011000	18	24	ERASE LINE (shift/left arrow)
00011001	19	25	(shifted right arrow key)
00011010	1A	26	(shifted down arrow key)
00011011	1B	27	(shifted up arrow key)
00011100	1C	28	HOME CURSOR
00011101	1D	29	
00011110	1E	30	
00011111	1F	31	CLEAR
00100000	20	32	SPACE (start of char. set)
00100001	21	33	!
00100010	22	34	"
00100011	23	35	#
00100100	24	36	$
00100101	25	37	%
00100110	26	38	&
00100111	27	39	'
00101000	28	40	(
00101001	29	41	)
00101010	2A	42	*
00101011	2B	43	+
00101100	2C	44	,
00101101	2D	45	-
00101110	2E	46	.
00101111	2F	47	/
00110000	30	48	0
00110001	31	49	1
00110010	32	50	2
00110011	33	51	3
00110100	34	52	4
00110101	35	53	5
00110110	36	54	6
00110111	37	55	7
00111000	38	56	8
00111001	39	57	9
00111010	3A	58	:
00111011	3B	59	;
00111100	3C	60	<
00111101	3D	61	=
00111110	3E	62	>
00111111	3F	63	?
01000000	40	64	@
01000001	41	65	A
01000010	42	66	B
01000011	43	67	C
01000100	44	68	D
01000101	45	69	E
01000110	46	70	F
01000111	47	71	G
01001000	48	72	H
01001001	49	73	I

01001010	4A	74	J
01001011	4B	75	K
01001100	4C	76	L
01001101	4D	77	M
01001110	4E	78	N
01001111	4F	79	O
01010000	50	80	P
01010001	51	81	Q
01010010	52	82	R
01010011	53	83	S
01010100	54	84	T
01010101	55	85	U
01010110	56	86	V
01010111	57	87	W
01011000	58	88	X
01011001	59	89	Y
01011010	5A	90	Z
01011011	5B	91	UP ARROW CHAR.
01011100	5C	92	DOWN ARROW CHAR.
01011101	5D	93	LEFT ARROW CHAR.
01011110	5E	94	RIGHT ARROW CHAR.
01011111	5F	95	CURSOR CHAR.
01100000	60	96	(shifted @ key)
01100001	61	97	a
01100010	62	98	b
01100011	63	99	c
01100100	64	100	d
01100101	65	101	e
01100110	66	102	f
01100111	67	103	g
01101000	68	104	h
01101001	69	105	i
01101010	6A	106	j
01101011	6B	107	k
01101100	6C	108	l
01101101	6D	109	m
01101110	6E	110	n
01101111	6F	111	o
01110000	70	112	p
01110001	71	113	q
01110010	72	114	r
01110011	73	115	s
01110100	74	116	t
01110101	75	117	u
01110110	76	118	v
01110111	77	119	w
01111000	78	120	x
01111001	79	121	y
01111010	7A	122	z
01111011	7B	123	
01111100	7C	124	

01111101	7D	125	
01111110	7E	126	
01111111	7F	127	
10000000	80	128	END
10000001	81	129	FOR
10000010	82	130	RESET
10000011	83	131	SET
10000100	84	132	CLS
10000101	85	133	CMD
10000110	86	134	RANDOM
10000111	87	135	NEXT
10001000	88	136	DATA
10001001	89	137	INPUT
10001010	8A	138	DIM
10001011	8B	139	READ
10001100	8C	140	LET
10001101	8D	141	GOTO
10001110	8E	142	RUN
10001111	8F	143	IF
10010000	90	144	RESTORE
10010001	91	145	GOSUB
10010010	92	146	RETURN
10010011	93	147	REM
10010100	94	148	STOP
10010101	95	149	ELSE
10010110	96	150	TRON
10010111	97	151	TROFF
10011000	98	152	DEFSTR
10011001	99	153	DEFINT
10011010	9A	154	DEFSNG
10011011	9B	155	DEFDBL
10011100	9C	156	LINE
10011101	9D	157	EDIT
10011110	9E	158	ERROR
10011111	9F	159	RESUME
10100000	A0	160	OUT
10100001	Al	161	ON
10100010	A2	162	OPEN
10100011	A3	163	FIELD
10100100	A4	164	GET
10100101	A5	165	PUT
10100110	A6	166	CLOSE
10100111	A7	167	LOAD
10101000	A8	168	MERGE
10101001	A9	169	NAME
10101010	AA	170	KILL
10101011	AB	171	LSET
10101100	AC	172	RSET
10101101	AD	173	SAVE
10101110	AE	174	SYSTEM

10101111	AF	175	LPRINT
10110000	B0	176	DEF
10110001	B1	177	POKE
10110010	B2	178	PRINT
10110011	B3	179	CONT
10110100	B4	180	LIST
10110101	B5	181	LLIST
10110110	B6	182	DELETE
10110111	B7	183	AUTO
10111000	B8	184	CLEAR
10111001	B9	185	CLOAD
10111010	BA	186	CSAVE
10111011	BB	187	NEW
10111100	BC	188	TAB(
10111101	BD	189	TO
10111110	BE	190	FN
10111111	BF	191	USING

BINARY	HEX.	DEC.	TAB.	BASIC COMMAND.
11000000	C0	192	0	VARPTR
11000001	C1	193	1	USR
11000010	C2	194	2	ERL
11000011	C3	195	3	ERR
11000100	C4	196	4	STRING$
11000101	C5	197	5	INSTR
11000110	C6	198	6	POINT
11000111	C7	199	7	TIME$
11001000	C8	200	8	MEM
11001001	C9	201	9	INKEY$
11001010	CA	202	10	THEN
11001011	CB	203	11	NOT
11001100	CC	204	12	STEP
11001101	CD	205	13	+
11001110	CE	206	14	-
11001111	CF	207	15	*
11010000	D0	208	16	/
11010001	D1	209	17	(Up Arrow)
11010010	D2	210	18	AND
11010011	D3	211	19	OR
11010100	D4	212	20	>
11010101	D5	213	21	=
11010110	D6	214	22	<
11010111	D7	215	23	SGN
11011000	D8	216	24	INT
11011001	D9	217	25	ABS
11011010	DA	218	26	FRE
11011011	DB	219	27	INP
11011100	DC	220	28	P0S

11011101	DD	221	29	SQR
11011110	DE	222	30	RND
11011111	DF	223	31	LOG
11100000	E0	224	32	EXP
11100001	E4	225	33	COS
11100010	E2	226	34	SIN
11100011	E3	227	35	TAN
11100100	E4	228	36	ATN
11100101	E5	229	37	PEEK
11100110	E6	230	38	CVI
11100111	E7	231	39	CVS
11101000	E8	232	40	CVD
11101001	E9	233	41	EOF
11101010	EA	234	42	LOC
11101011	EB	235	43	LOF
11101100	EC	236	44	MKI$
11101101	ED	237	45	MKS$
11101110	EE	238	46	MKD$
11101111	EF	239	47	CINT
11110000	F0	240	48	CSNG
11110001	Fl	241	49	CDBL
11110010	F2	242	50	FIX
11110011	F3	243	51	LEN
11110100	F4	244	52	STR$
11110101	F5	245	53	VAL
11110110	F6	246	54	ASC
11110111	F7	247	55	CHR$
11111000	F8	248	56	LEFT$
11111001	F9	249	57	RIGHT$
11111010	FA	250	58	MID$
11111011	FB	251	59	' (REM)
11111100	FC	252	60	
11111101	FD	253	61	
11111110	FE	254	62	
11111111	FF	255	63	

* *

PRODUCED IN AUSTRALIA
by

MICRO-80 PRODUCTS

P.O. BOX 213 GOODWOOD SA 5034