SOFTWARE

TECHNICAL

MANUAL

PRESENTED BY

HOUSTON

MICRO

COMPUTER

TECHNOLOGIES

5313 BISSONNET
BELLAIRE, TEXAS 77401
(713) 661-2005

DISCLAIMER

Houston Micro-Computer Technologies, Inc. in the presentation of the Software Technical Manual, is in no way attempting to usurp the information contained in the Level II Basic Manual provided by Microsoft. It is our desire only, to clarify some of the features available in Level II Basic.

COPYRIGHT

5313 Bissonet
Bellaire, Texas 77401
713-661-2005

TABLE OF CONTENTS

DATA CONVERSIONS AND I/O CONT'D

1. INTRODUCTION

The purpose of this manual is to provide the assembly programmer with documentation of the TRS-80 Level II BASIC ROM entry points and provide working examples of their use. Several implicit assumptions were made about the needs and background of the readers. First, it is assumed that the reader understands and programs in Z80-assembly language. Second, it is assumed that the assembly programmer is primarily interested in writing fast, computationally oriented programs. These assumptions have influenced the material included in this manual. This manual is not an exhaustive technical program manual. Rather, it is a brief, concise description of computationally oriented routines in the ROM. Level II BASIC and DISK BASIC functions and commands which are not discussed in the text are referenced in the memory maps of Appendix A. The entry points discussed in this manual are summarized in Tables 2 through 6 and Table 8. It is worthwhile to reproduce these tables so they may be kept handy for reference.

This manual is organized in sections which emphasize different aspects of computation. Section 2 discusses data handling. This includes general information about the Level II ROM, notational conventions and descriptions of routines which move data. Section 3 is concerned with compares. Compares act in the same sense as the Z80 CP (compare) instruction. Flags are set as if the data values had been subtracted. Section 4 discusses entry points for the arithmetic operation (+,-,* and /) for integer, single and double precision arithmetic. Section 5 is concerned with the BASIC mathematical functions such as LOG, SQR, SIN ... as well as the number type conversion routines, CINT, CSNG and CDBL. Section 6 discusses data conversion routines (ASCII <-> internal numbers) and I/O routines, (keyboard, video, tape, graphics, and line printer). Finally, Section 7 discusses logic, which extends the USR option. By using variable location, variable evaluation and expression evaluation logic it is possible to allow an arbitrary number of user exits and an arbitrary number of user arguments.

The appendices contain enhancing information. Appendix A contains complete maps of Level II BASIC and DISK BASIC keyword entry points. Appendix B discusses detailed tape I/O functions and square wave music generation. Appendix C presents detailed tape formats and Appendix D presents a test and demonstration program.

Finally, a word about RST instructions. These instructions pass control to RAM link areas. This manual assumes these link areas have been initialized by Level II BASIC. Caution should be used when operating under DISK BASIC.

2. DATA HANDLING

This section is concerned with the storage format and manipulation of integers, single precision floating point, (or simply single), and double precision floating point data, (double). Also, two calls can handle strings up to 256 bytes in length.

First, some general aspects of data handling are presented. The Level II ROM never uses the alternative register set. By using EX, AF, AF' and EXX instructions, registers can be saved during ROM calls. Similarly, the IX and IY register are preserved by Level II ROM subroutines. The Level II ROM does use RAM storage. Both stack memory and pre-assigned RAM are used. RAM through 42E8H should be considered unavailable for user programs. When the SYSTEM command is used to initiate execution of programs, the stack is initialized in the I/O buffer area, (SP <-- 4288H). The user should redefine the stack before ROM subroutines are called.

Certain RAM locations are used extensively by the arithmetic routines. The most frequently accessed memory acts as an accumulator, and will be referred to as ACC, (see Table 1). The ACC notation will be used to refer to RAM storage just as A is used to refer to the register storage in the Z80 CPU. Most ROM routines expect the ACC to contain a specific type of data, (integer, single or double precision). For routines that allow different input data types, the type is indicated by the contents of address 40AFH. This will be abbreviated NTF (number type flag). The type conventions are:

```
NTF = (40AFH) = 2  Integer
                3  String
                4  Single precision
                8  Double precision
```

Note that the NTF corresponds to the length of the data. The exception is string data where, in most applications, the ACC contains a pointer to the string length (1 byte) and string address (2 bytes). The RAM storage labled DTEM in Table 1 is used primarily by the double precision arithmetic operations. The ACC and DTEM are changed by data conversion and string operations. DTEM is not changed by compares, integer or single precision arithmetic operations or arithmetic functions. Finally, unless otherwise indicated, register values can be considered as having been modified by ROM calls.

Table 2 describes the data handling routines. The indirect addresses are indexed upward during the move. Data loaded to the ACC, DTEM or the BCDE registers are correctly arranged for arithmetic operations provided the originating memory was correctly arranged, (see Level II Reference manual pages 8/8 - 8/10).

TABLE 1 ACC & DTEM DATA ORGANIZATION

	ADDRESS	DOUBLE	SINGLE	INTEGER
ACC	411DH	LSB		
	--	---		
	4121H	---	LSB	LSB
	4122H	---	---	MSB
	4123H	MSB	MSB	
	4124H	EXP	EXP	
DTEM	4127H	LSB	LSB	LSB
	4128H	---	---	MSB
	4129H	---	MSB	MSB
	412AH	---	EXP	
	---	---		
	412DH	MSB		
	412EH	EXP		

See the Level II BASIC Reference Manual, pages 8/9 and 8/10 for syntax of single and double precision numbers.

TABLE 2 DATA HANDLING

Unmentioned register contents are changed.

CALL	COUNT	TO	FROM	DISPOSITIONS
09D7H	B (0 => 256)	(HL)	(DE)	DE <-- DE+A HL <-- HL+A
09D6H	A (0 => 256)	(HL)	(DE)	DE <--DE+A HL <--HL+A
09D3H	NTF	(HL)	(EE)	DE <-- DE+NTF HL <-- HL+NTF
09D2H	NTF	(DE)	(HL)	DE <-- DE+NTF HL <-- HL+NTF
09CEH	4	(HL)	(DE)	DE <-- DE+4 HL <-- HL+4
09CBH	4	(HL)	ACC	HL <-- HL+4
09B1H	4	ACC	(HL)	HL <-- HL+4
09B4H	--	ACC	BCDE	BC,HL unchanged
09BFH	--	BCDE	ACC	------
09C2H	4	BCDE	(HL)	HL <-- HL+4
0A9AH	--	ACC	HL	NTF <-- 02 BC,DE unchanged
09F4H	NTF	ACC	DTEM	------
09FCH	NTF	DTEM	ACC	------
09A4H	4	Stack*	ACC	A,BC,HL unchanged

*To retrieve the value, use POP BC followed by POP DE in the calling program. If the accumulator was an integer, DE will be the integer value.

3. COMPARES

This section is concerned with logic which compares numeric and character values. The logic is designed to be similar in philosophy to the compare (CP) instruction. That is, the Z and S flags are set to express the result of a subtraction, but the subtraction is never actually performed and the original values compared remain unchanged. Table 3 summarizes the compare logic.

The first compare can be executed by CALL 1C90H or by RST 18H. This compare sets or resets the Z flag as if the subtraction HL-DE had been performed. HL and DE are treated as unsigned (positive) integer constants. This logic is best suited for use where HL and DE are addresses.

The compares 2) through 6) are self-explanatory. They all set the Z and S flags as a result of the indicated subtraction, (compare 3 can be considered ACC-0).

Compare 3) is related to the SGN function (see section 5). Using it with NTF = 3 is an error, in which case the machine is reinitialized into BASIC. All of these compares set the A register to FFH, 0 or 1 provided the subtraction would result a negative, zero or positive respectively.

Compare 7) can be executed by CALL 25D9H or RST 20H (in Level II BASIC). This compare test the NTF (Number Type Flag) as indicated.

RST 8 and RST 10H, (see compares 8 and 9 of Table 3), are used in scanning strings, mostly in conjunction with the BASIC interpreter. First, the RST 10H logic is displayed in Figure 1. This logic begins by incrementing HL and checking characters at (HL). It increments through spaces, (" "), and control characters with value 9 and 10 (carriage returns). The routine returns C set when A = (HL) contains a numeric ASCII character. Z is set for A= (HL) = Zero (BASIC end-of-line) or ":" (BASIC end of statement).

RST 8 compares the character pointed to by HL, with the character located at the return address, ((SP)). If they are not equal, a SN ERROR results and the machine is reinitialized into BASIC. If they are equal the return address is incremented (so the return by-passes the test character) and control is passed to the RST 10H Logic. The RST 8 logic is used by BASIC to check for expected characters, such as "(" in SIN(X), and then find the next non-blank character, (the RST 10H logic).

TABLE 3 COMPARE LOGIC

	CALL	COMPARE	NOTES*
1)	1C90H (RST 18H)	HL-DE (unsigned)	A used. S and C flags not well defined.
2)	0A39H	HL-DE (signed)	A = 00 if = 0 A = 01 if > 0 A = FF if < 1
3)	0994H	ACC (integer, single double)	A as from 0A39H. Error if NTF = 3 (string).
4)	0A0CH	ACC-BCDE (single)	A as from 0A39H.
5)	0A78H	DTEM-ACC (double)	A as from 0A39H. Uses all registers.
6)	0A4FH	ACC-DTEM (double)	A as from 0A39H. Uses all registers.
7)	25D9H (RST 20H)	NTF-3	A = NTF-3 Z & S set accordingly C set if NTF isn't =8 C is reset if NTF =8
8)	1C96H (RST 8)	(HL)-((SP))	See discussion. SN Error if not =, else go 1D78H, (see below). BC, DE, unchanged.
9)	1D78H (RST 10H)	(HL)	See discussion. BC, DE unchanged. HL incremented. C set for numeric.

* No registers are modified except as indicated. Also, no RAM locations are modified. Test 1-6 set or reset Z & S accordingly. Test 2-6 set or reset C to S.

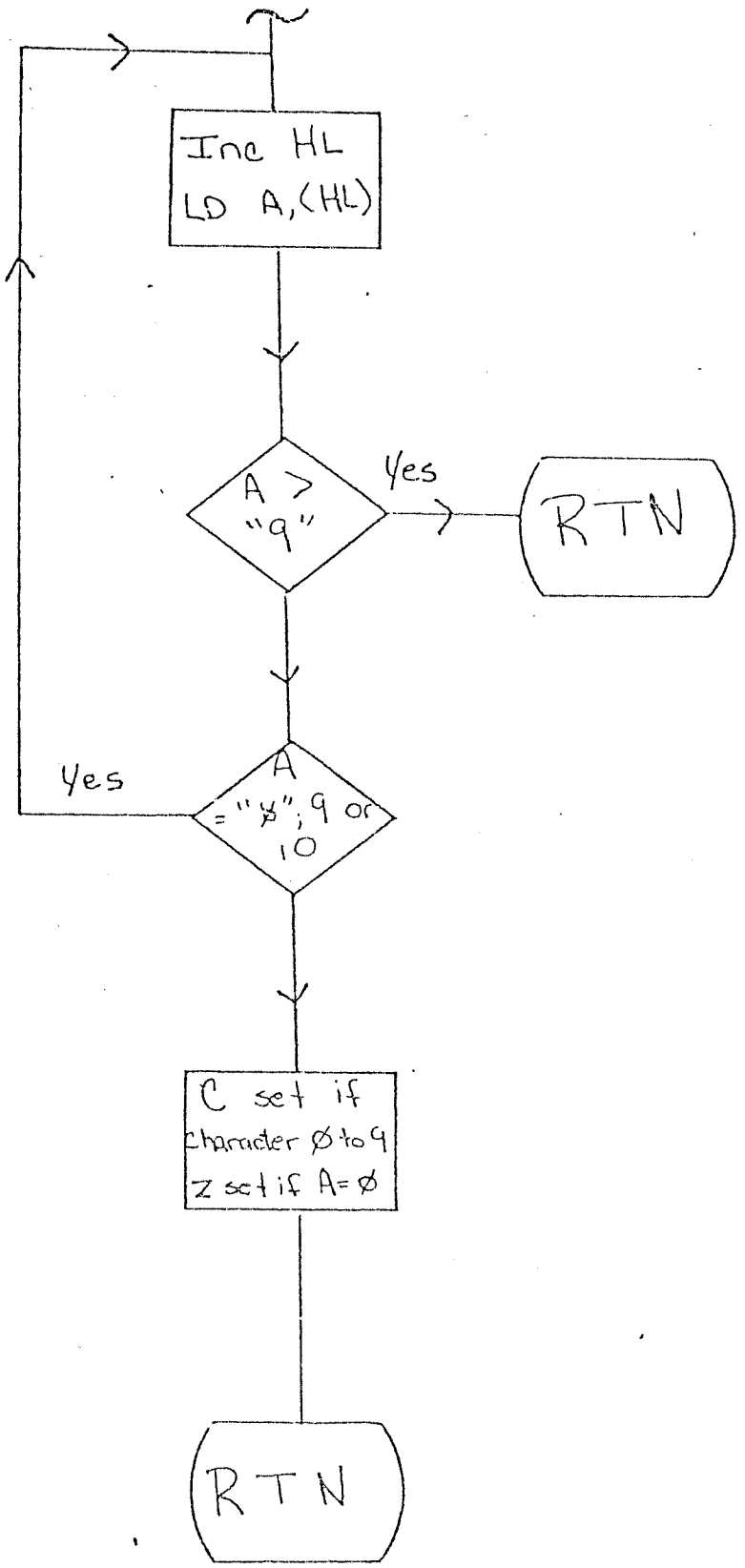


Figure 1

CALL 1D78H, (or RST 10H in Level II BASIC).

This logic reads through spaces and control characters 9 and 10 to return C set for numeric characters.

4. ARITHMETIC

Calling arithmetic routines is straight forward, as depicted in Table 4. It is up to the user to make sure that the input agrees in number type and that the correct routine is called. If conversions from one type to another are necessary, refer to Section 5 for functions CINT, CSNG or CDBL.

Input to the arithmetic computations can be regarded as unpredictable after the CALL is executed. The results of each computation are stored in the ACC, and agree in number type with the input. Exceptions to this are integer divide,(/), which always returns single float, and integer +, - and * which return single float upon overflow. Check the NTF to see if conversion to single has occurred.

If single or double exponent overflow occurs, or zero divide occurs, an error condition will result in the reinitialization into BASIC. Finally, the routines use stack space which must be provided by the user.

TABLE 4 ARITHMETIC OPERATIONS

Operators (op) = +,-,*,/

	+	-	*	/
INTEGER DE op HL	0BD2H	0BC7H	0BF2H	2490H
	(single out to ACC on overflow)			(single out)
SINGLE BCDE op ACC	0716H	0713H	0847H	08A2H
DOUBLE ACC op DTEM	0C77H	0C70H	0DA1H	0DE5H

All output is returned in the ACC. Integer +, -, and * output to ACC and HL, and NTF <-- 2. Integer overflow and integer divide, /, result in single float to ACC with NTF <-- 4.

5. BASIC FUNCTIONS

This section presents the Level II BASIC arithmetic function entry points, (See Table 5). Other BASIC keyboard entry points will be discussed later or can be found on the memory map, (Appendix A). Complete descriptions of the arithmetic functions can be found in the Level II BASIC Reference Manual, Chapter 7. The user must be careful to supply arguments within the allowed range or an error will occur and the machine will be initialized into BASIC. Also, it is necessary for the user to supply the correct number type since many of the routines do not check the NTF.

The functions CINT, CSNG and CDBL are useful in converting number types for use in arithmetic operations or other functions.

Also included in Table 5 are routines which change the sign of integer or float numbers,(chg.sgn.).

TABLE 5 LEVEL II BASIC FUNCTIONS

FUNCTION	CALL	OUTPUT	INPUT
LOG	0809H	ACC single	ACC single
SQR	13E7H	"	"
EXP	1439H	"	"
COS	1541H	"	"
SIN	1547H	"	"
TAN	15A8H	"	"
ATAN	15BDH	"	"
ABS	0977H	Same as input ACC,NTF=2,4 or 8	
CINT	0A7FH	HL,ACC, NTF=2	"
CSNG	0AB1H	ACC, NTF=4	"
CDBL	0ADBH	ACC, NTF=8	"
FIX	0B26H	ACC, NTF=2 or 4*	"
INT	0B37H	"	"
RND	14C9H	ACC, NTF=4	"
SGN	098AH	ACC= -1,0,1,NTF=2	"
chg.sgn.	0C51H	HL,ACC,NTF=2	HL
chg.sgn.	0982H	ACC <-- -ACC ACC,NTF=4 or 8 BCDE unchanged.	

*NTF = 2 if amplitude < 32767.

6. DATA CONVERSIONS AND I/O

This section is concerned with the conversion of numbers to strings (and vice versa) and I/O to external devices. It is divided into four parts; data conversion, keyboard input, tape I/O, general output (tape, video, and line printer), and graphics. Additional detail on cassette I/O, along with cassette tape formats can be found in the appendicies. A summary of the calls discussed below can be found in Table 6.

Data Conversion

CALL 0FBDH converts the ACC, NTF= 2, 4 or 8 into an ASC II character string whose starting address, HL, is about 4130H. In performing the conversion, the contents of the ACC and DTEM are destroyed. The ASC II string is terminated with a zero byte.

CALL's 0E65H and 0E6CH convert an ASC II character string into a number stored in the ACC and fixes the NTF, (DTEM is destroyed). CALL 0E65H stores such that NTF=8 and CALL 0E6CH stores the number type with the least length compatible to the input string. The rules for the string formats are those used in Level II BASIC. The ASC II character string converted by these routines must be terminated by a comma or zero byte. (zero bytes are inserted by keyboard input routines 0361H and 1BB3H discussed below).

These data conversion routines suggest an easy subroutine to input single or double precision immediate values to the ACC and NTF, (see Example 1). Notice how the subroutine retrieves the string address from the return address. Eventually, control is returned to the statement after the DEFM.

EXAMPLE 1 Loading Immediate Values

```
                .
                .
                .
        CALL      LOAD
        DEFM    '1.23456,'    ;immediate value
                .
                .
                .
LOAD    POP       HL
        CALL      0E65H     ; convert string to ACC + NTF
        INC       HL
        PUSH      HL
        RET
```

TABLE 6 DATA CONVERSIONS AND I/O

CALL	OUTPUT	INPUT
Data conversion		
0FBDH	HL <-- string address	ACC,NTF = 2,4 or 8
0E65H	ACC,NTF = 8. HL <-- delimiter address	HL <-- string address. String terminated with zero byte or comma
0E6CH	ACC, NTF = 2,4,or 8. HL <-- delimiter address	HL <-- string address. String terminated with zero byte or comma.
Keyboard Input		
05D9H	HL unchanged. B input length. Input to video.	HL Buffer address. B Buffer length.
0361H (see warning)	HL = (40A7H)-1 input string terminated with zero.	(40A7H) Buffer address. F0H Buffer length.
1BB3H (see warning)	Outputs "? " with CALL 032AH and then jumps to 0361H.	
002BH	A = ASC II of keystroke A = 0 no keystroke	
0049H	A = ASC II of keystroke (waits for keystroke)	
Tape Input Logic		
0212H	Use A as cassette select latch and store it at 37E4H, (A = 0 for cassette #-1 and A = 255 for cassette #-2). 0212H also turns on cassette. BC, DE & HL unchanged.	
0296H	Read until sync byte, A5, H, found. BC,DE & HL unchanged.	
0235H	Read next byte into A. BC, DE & HL unchanged.	
01F8H	Turn off cassette. BC, DE & HL unchanged.	

Tape Output Logic

0212H	-see Tape Input Logic-
0287H	Write leader and A5H sync byte. DE and HL unchanged.
0264H	Write byte in A to tape. BC, DE and HL unchanged.
01F8H	-see Tape Input Logic-

General Output

28A7H	HL string address (delimiter 00 or 22H byte) (409CH) = -1,0,1 for tape, video, or line printer
032AH	Displays byte in A. (409CH) -1,0,1 for tape, video, or line printer
0033H	Display byte in A to video.

Graphics

GRAPH
(See Ex. 7)

Input: B = X coordinate; 0-127
A = Y coordinate; 0-47
H = 00H for POINT
80H for SET
01H for RESET

Output; If POINT then ACC = 0000H for off
= FFFFH for on

The most powerful routines are at 05D9H, 0361H and 1BB3H (See Table 6). The most fundamental of these routines is at 05D9H. This routine accepts keyboard input as if INPUT were being executed in BASIC. The keystrokes are reflected on the video. The buffer address is input by HL and buffer length is input by register B. Keyboard input exceeding the buffer length is ignored. After a carriage return, control passes to the calling program with HL the original buffer address and B the length of the input string.

The next routine is 0361H. This routine uses the buffer address stored at 40A7H, HL <-- (40A7H), and fixes buffer length, B <-- F0H. A CALL 05D9H is then executed (see above). Upon return, the input string is terminated with a zero byte, (necessary to convert ASCII --> ACC).

Warning! In Level II BASIC,
the buffer area is 41E8H=(40A7H). If you initiate your program via SYSTEM, the stack pointer is set SP <-- 4288H. So please reset the SP out of the buffer area or redefine (40A7H) before input.
Warning! In DOS BASIC,
CALL 0361H will CALL DOS code via address 41AFH. The purpose of this call is unknown and may affect performance. To overcome, use Level II or set (41AFH) = C9H, (RET).

Finally, the most useful routine is 1BB3H. This routine displays the prompt "? " and passes control to 0361H.

An example of keyboard input and conversion is most useful at this point. Suppose it is necessary to input two single precision variables, analogous to:

```
INPUT X,Y
```

Assembler code to do this is given by Example 2.

Note the use of RST 10H to skip past the delimiter and find the next non-blank character. The CALL 0E69H is quite forgiving about what character is used as a delimiter. Any given character that does not make sense as a part of the number causes the ACC to be fixed, and control to be returned to the user. Also, recall that after RST 10H, the user can check the C (carry) flag to see if the next character is numeric.

The last keyboard scan routines discussed, returns only one byte in register A. CALL 002BH returns the ASCII representation of the current key being pressed. A=0 indicates the keyboard is clear. CALL 0049H does the same, but waits until A does not equal 0. It is possible to disassemble the code beginning at 2BH and learn how to intercept the keyboard driver routine via its address in the Keyboard DCB (see page D/1 of the Level II BASIC Reference Manual). By putting a delay routine in the intercepting program, keyboard debouncing can be achieved. Using the delay routine at 60H (see Appenidix B) gives the ASM program in Example 3. Note that if the system is reinitialized, MEMORY SIZE? appears, then the debounce routine must be reloaded.

EXAMPLE 2 Input X,Y in Assembler

Redefine SP out of buffer area if using SYSTEM.

```
    CALL    1BB3H   ; prompt and Keyboard input
    RST     10H     ; locate 1-st character (see Section 3)
    CALL    0E6CH   ; convert X to ACC.
    PUSH    HL      ; save string pointer
    CALL    0AB1H   ; CSNG
    LD      HL,X    ; destination address
    CALL    09CBH   ; save X, (see Section 2)
    POP     HL      ; restore string pointer
    RST     10H     ; locate next character.
    CALL    0E6CH   ; convert Y to ACC
    CALL    0AB1H   ; CSNG
    LD      HL,Y    ; destination address
    CALL    09CBH   ; save Y
    .
    .
    .
X   DEFS    4
Y   DEFS    4
    END
```

EXAMPLE 3 Keyboard Debouncing

```
        ORG     4016H     ; location of keyboard driver address
        DEFW    DEBNCE    ; load new driver address
        ORG     7E00H     ; high memory
DEBNCE  CALL    03E3H     ; call keyboard driver.
        LD      H,A       ; save keyboard input
        LD      BC,02AFH  ; count for 1/100 second delay
        CALL    60H       ; delay loop
        LD      A,H       ; restore keyboard input
        RET
        END
```

It may also be useful to understand how the keyboard RAM is used. The keyboard forms a matrix. When a key is pressed, the row and column of the matrix determine memory address and memory contents, respectively. From a programming view, one can see what happens by running the program of Example 4. (Be sure to hold down the keys during the PRINT.) Also, if PEEK (14463)=0, then no keys are being pressed, including SHIFT. To exclude SHIFT, use PEEK (14591).

EXAMPLE 4 Displays Keyboard Memory

```
10 FOR I=0 TO 7

20 AD = 14336 + 2 I

30 PRINT AD,PEEK(AD)

40 NEXT I

50 PRINT

60 IF INKEY$="" THEN 60 ELSE 10
```

EXAMPLE 5 Detecting Multi-Keying Entries

Using this knowledge, it is possible to detect multiple key entries. For example, the following program will draw verticle, horizontal or diagonal lines using the keys for right arrow, left arrow, down arrow, and up arrow.

```
10 X=64: Y=24:

20 I=PEEK(14400): IF I=0 THEN 20

25 DX=0: DY=0

30 IF I AND 8 THEN DY =-1

40 IF I AND 16 THEN DY=1

50 IF I AND 32 THEN DX=-1

60 IF I AND 64 THEN DX=1

70 X=X+DX: Y=Y+DY: SET(X,Y)

80 GOTO 20
```

In particular, note that the use of "AND" allows for the determination of multiple key entries.

Tape I/O

Entry points for the tape I/O logic are given in Table 6. The use of these entry points is straightforward and no additional detail will be given here. Appendix B discusses how to drive the I/O port to acheive square wave music. Example 6 demonstrates the use of the tape I/O calls to accomplish tape copies. Appendix C contains tape formats for tapes written with CSAVE, PRINT#, EDTASM W and EDTASM A commands.

Note that the tape I/O requires the user to know when to stop reading. That is,CALL 0235H may load garbage into the A register after the end of valid data or, CALL 0235H may never return control to the user. Also data is transmitted bit by bit. Each bit is preceeded by a timing bit. Thus, it is acceptable to introduce appreciable delays in the output. This would allow computation to be performed during output or input. Also note that no interrupts are allowed during tape I/O, so turn off the clock!

EXAMPLE 6 Tape Copy Program

```
00010          ORG     4C90H       ;19600
00020 READ     LD      A,0
00030          CALL    0212H       ; DEFINE DRIVE
00040          CALL    0296H       ; FIND SYNC BYTE
00050          LD      HL,4CE0H    ; DEST ADDR 19680
00060          LD      BC,331FH    ; # BYTES I/O 13087
00090  L1      CALL    0235H       ; READ ONE BYTE
00120          LD      (HL),A
00130          INC     HL
00140          DEC     BC
00150          LD      A,B
00160          OR      C
00170          JR      NZ,L1
00180          CALL    01F8H       ; DISABLE DRIVE
00190          JP      1A19H       ; RETURN TO BASIC
00200 WRITE    LD      A,0         ; 19630
00210          CALL    0212H       ; DEFINE DRIVE
00220          CALL    0287H       ; WRITE LEADER
00230          LD      HL,4CE0H    ; ORIG ADDR
00240          LD      BC,331FH    ; # BYTES I/O 13087
00250  L2      LD      A,(HL)
00280          CALL    0264H       ; OUTPUT BYTE
00310          INC     HL
00320          DEC     BC
00330          LD      A,B
00340          OR      C
00350          JR      NZ,L2
00360          CALL    01F8H       ; DISABLE DRIVE
00370          JP      1A19H       ; RETURN TO BASIC
00380          END
```

Under System /19600 will cause 13087 bytes to be read to RAM (Reset will stop the program). /19630 will cause the same 13087 bytes to be written to tape.

General Output

CALL 28A7H provides the most general purpose output routine. An example of its use is given in Appendix D. This routine will output a character string to the cassette, video or line printer, depending on the contents of 409CH. The address of the character string is input with HL, and the string must be terminated with a zero or 22H (quote) delimiter. Upon return, for video, 40A6H contains the current cursor position on the line. When using video, the control characters given by the Level II BASIC Reference Manual, pp. C/1, will be effective. Line printer control characters are given on page 10/3. When used with tape, CALL 28A7H only replaces Call 0264H. It is still up to the user to CALL 0212H, 0287H and 01F8H appropriately. CALL 28A7H does not transmit the string delimiter character.

CALL 032AH performs the same function as CALL 28A7H, except that only the character stored in the A register is transmitted. CALL 0033H is the same as 032AH except it is for video only and does not store the line cursor position in 40A6H.

Graphics

Use of the graphic logic is complicated by the fact that the logic deciphers the BASIC argument list. This is best overcome by setting up parameters as indicated in Table 6 and executing CALL GRAPH where GRAPH as is given in Example 7. This approach bypasses the tests for bounds on the X and Y coordinates. Thus it is up to the user to be sure they are within 0-127 and 0-47 respectively. The routine GRAPH sets up HL to point to a dummy string, ");" to satisfy a RST 8 at the end of the graphics logic.

EXAMPLE 7 Graph Subroutine

```
10          ORG         07000H
20 ENTRY    CALL        01C9H      ;CLS
30          LD          B,40       ;INITIALIZE X COORDINATE
40 LOOP     LD          A,B        ;FIX Y COORDINATE
50          PUSH        BC         ; SAVE X COORDINATE
60          LD          H,80H      ;PERFORM SET FUNCTION
70          CALL        GRAPH
80          POP         BC         ;RECALL X COORDINATE
90          DJNZ        LOOP       ;DECREMENT X COORD. &
                                    REPEAT
100 STOP    JR          STOP
110 ;
120 ;
130 GRAPH   PUSH        HL         ;PUSH INDICATOR
140         PUSH        BC         ;PUSH X COORDINATE
150         LD          HL,LGRPX
160         JP          0150H
170 LGRPX   DEFW        3B29H      ;STRING");"
180         END         ENTRY
(This program plots a diagonal line from (40,40) to (1,1))
```

7. EXTENDING THE USR OPTION

It is easy to provide additional flexability in the USR option. For an example of what the possibilities are, consider a program which must compute

C <-- A * (B+A)

where A,B,and C are matrices (see Level II Reference Manual p. 6/4-6). The program could be coded in BASIC as follows:

```
100 DUMMY=USR ADD (B,A) 'B = B+A
110 DUMMY=USR MULT (C,A,B) 'C = A*B
```

Specifically the possibilities are, (1) to allow any number of arbitrarily named user exits, (2) to allow any number of arguments in the argument list, and (3) to pass arguments by address, (hence the value of the argument can be changed by the object program). This section will present a program which allows this to be done, (see Example 8, the DEFUSR program). Also, various alternatives will be discussed. The DEFUSR program will not be discussed in detail except to note that the data loaded between 4150H and 41BDH is destroyed when the machine is initialized. Hence, if MEMORY SIZE? appears, DEFUSR must be reloaded. Also, DEFUSR can only be used with Level II BASIC since it interfers with the ROM-DISK-BASIC linkage, (see Appendix A).

Here is how to use DEFUSR. Before a user program is called, its location in memory must be defined with the DEF statement.

The syntax of the DEF statement is:

DEF Name = Address

For example:
```
DEF MULT = 30000
DEF ADD = 14000 * 2
```

Names are arbitrary length alpha-numeric strings terminating with space or "=". DEF need only be used once for each name. Names can be defined any number of times and only the latest definition is effective. For addresses above 32767, use:

desired address - 65536

The only restrictions with DEF are that the names and their addresses are stored in a table one hundred bytes long with no overflow protection (see statement 140).

EXAMPLE 8 THE DEFUSR PROGRAM

```
00010           ORG     4188H   ;DISC-BASIC LINK IN RUN INIT.
00020           JP      RESET   ;GO TO RESET TABLE
00030           ORG     4158H   ;DISK-BASIC LINK ON DEF
00040           JP      DEF
00050           ORG     41A9H   ;DISC-BASIC LINK ON USR
00060           JP      USR
00070           ORG     7EE0H   ;32480
00080 ;
00090 ;RESET TABLE TO NO ENTRIES
00100 ;
00110 RESET     LD      IX,TABLE+1      ;ADDRESS OF TABLE+1
00120           LD      (TNEXT),IX      ;POINTER FOR NEXT ENTRY
00130           RET
00140 TABLE     DEFB    0
00150           DEFS    100     ;TABLE
00160 TNEXT     DEFW    TABLE   ;NEXT VACANT ENTRY.
00170 ;
00180 ;UPDATE TABLE FROM DEF COMMAND
00190 ;
00200 DEF       CALL    SCAN    ;LOOK UP NAME IN TABLE.
00210           PUSH    AF      ;SAVE FLAG
00220           LD      B,C     ;BASIC STRING LENGTH
00230           EX      DE,HL   ;DE GETS BASIC, HL GETS TABLE
00240           CALL    09D7H   ;MOVE STRING FROM BASIC TO TABLE
00250           EX      DE,HL   ;HL GETS BASIC, DE GETS TABLE
00260           PUSH    DE      ;PUSH TABLE ADDRESS POINTER
00270           DEC     HL
00280           RST     10H     ;SEARCH FOR "="
00290           RST     8       ;CHECK FOR "=" & FIND NEXT CHAR.
00300           DEFB    213     ;BASIC "="
00310           CALL    2337H   ;EVALUATE EXPRESSION.
00320           PUSH    HL      ;SAVE BASIC POINTER
00330           CALL    0A7FH   ;CINT TO HL
00340           EX      DE,HL   ;DE GETS USR ADDRESS
00350           POP     HL      ;BASIC POINTER
00360           POP     IX      ;TABLE ADDRESS POINTER
00370           LD      (IX),E  ;LOAD ADDRESS TO TABLE
00380           INC     IX
00390           LD      (IX),D
00400           INC     IX
00410           LD      (IX),0
00420           INC     IX
00430           POP     AF      ;FLAG FOR NO ENTRY FOUND
00440           OR      A
00450           RET     NZ      ;GO IF ENTRY WAS AN UPDATE
00460           LD      (TNEXT),IX      ;UPDATE TABLE END POINTER
00470           RET
00480 ;
00490 ;LOOK UP STRING AT HL IN TABLE, (STRING DELIMITER IS
00500 ;SPACE, "=" OR "(" ).
00510 ;IF FOUND:        DE POINTS TO TABLE NAME
00520 ;                 IX POINTS TO TABLE ADDRESS
00530 ;                 A = FFH
00540 ;IF NOT FOUND:    DE=(TNEXT)
00550 ;                 A = 0
00560 ;EITHER CASE:     C = STRING LENGTH
```

```
00570 ;                 HL UNCHANGED
00580 ;
00590 SCAN      DEC     HL          ;BACK UP POINTER
00600           RST     10H         ;FIND NEXT CHARACTER
00610           PUSH    HL          ;SAVE POINTER
00620           LD      C,0
00630 LBL1      INC     C
00640           INC     HL          ;NEXT CHARACTER.
00650           LD      A,(HL)
00660           CP      32
00670           JR      Z,LBL2      ;GO ON SPACE
00680           CP      213
00690           JR      Z,LBL2      ;GO ON BASIC "="
00700           CP      40
00710           JR      NZ,LBL1     ;GO NOT "("
00720 LBL2      POP     HL          ;RESTORE BASIC POINTER
00730           LD      DE,TABLE    ;1-ST TABLE ADDRESS
00740 NTBL      PUSH    HL          ;ORIGINAL HL
00750           PUSH    DE          ;NEXT TAABLE ADDRESS
00760           LD      B,C         ;BASIC STRING LENGTH
00770 ITER      LD      A,(DE)      ;LOAD TABLE CHARACTER
00780           CP      (HL)        ;COMPARE BASIC CHARACTER
00790           JR      NZ,NMTCH    ;GO NOT EQUAL
00800           INC     HL
00810           INC     DE
00820           DJNZ    ITER        ;NEXT CHARACTER
00830           PUSH    DE          ;POSSIBLE MATCH, CHECK TABLE
00840           POP     IX
00850           LD      A,(IX+2)
00860           OR      A           ;TEST (DE+2)
00870           JR      NZ,NMTCH    ;GO BAD TABLE ENTRY LENGTH
00880           POP     DE          ;TABLE NAME ADDRESS
00890           POP     HL          ;ORIGINAL BASIC POINTER
00900           LD      A,0FFH
00910           RET                 ;RETURN ON MATCH
00920 NMTCH     POP     DE          ;ADDRESS PREVIOUS TABLE ENTRY
00930 LBL3      LD      A,(DE)      ;LOOK FOR NEXT TABLE ENTRY
00940           INC     DE
00950           OR      A
00960           JR      NZ,LBL3     ;GO IF (DE-1) NOT ZERO.
00970           LD      HL,(TNEXT)
00980           RST     18H         ;COMPARE (TNEXT)-DE
00990           POP     HL          ;RESTORE BASIC POINTER
01000           JR      NZ,NTBL     ;NEXT TABLE ENTRY
01010           LD      A,0
01020           RET
01030 ;
01040 ;THIS LOGIC REEPLACES THE ROM USR CODE.
01050 ;IT LOOKS UP THE USR PROGRAM ADDRESS WITH CALL SCAN.
01060 ;IT THEN STUFFS THE ARGUEMENT ADDRESSES TO THAT PROGRAM
01070 ;ADDRESS (HENCE ACRUEMENTS CAN ONLY BE SIMPLE VARIABLES).
01080 ;A RETURN ADDRESS IS PUSHED AND CONTROL IS PASSED TO THE
01090 ;USR AT THE ADDRESS AFTER THE LAST ARGUEMENT ADDRESS.
01100 ;
01110 USR       POP     DE          ;WIPE OUT RETURN ADDRESS
01120           INC     HL
```

EXAMPLE 8 cont'd

```
01130           CALL    SCAN      ;GET USR ADDRESS
01140           OR      A
01150           JP      Z,1997H   ;NO TABLE ENTRY IS SN ERROR
01160           LD      E,(IX)
01170           LD      D,(IX+1)
01180           PUSH    DE
01190           POP     IX        ;USER ADDRESS
01200           LD      B,0
01210           ADD     HL,BC
01220           DEC     HL
01230           RST     10H       ;FIND "("
01240           RST     8         ;CHECK  "("
01250           DEFB    28H       ;"("
01260 NXTARG.   CALL    260DH     ;VARIABLE FINDING LOGIC
01270           LD      (IX),E    ;SAVE ARGUEMENT ADDRESS
01280           INC     IX
01290           LD      (IX),D
01300           INC     IX
01310           DEC     HL        ;FIND ","
01320           RST     10H
01330           CP      44
01340           JR      NZ,OUT    ;NO ","
01350           RST     10H
01360           JR      NXTARG    ;NEXT ARGUEMENT
01370 OUT       RST     8         ;CHECK FOR ")"
01380           DEFB    29H       ;")"
01390           PUSH    HL        ;PUSH POINTER
01400           LD      HL,0890H
01410           PUSH    HL        ;PUSH RETURN ADDRESS
01420           JP      (IX)      ;JUMP TO USER
01430           END
```

To call an assembler program, use the syntax

Variable = USR Name (argument variables)

As with the standard USR exit, Variable is assigned the value left in ACC and NTF by the user program. Using the previous DEF example,

X = USR MULT (C,A,B)

causes the USR program located at 30000 to be executed. Only simple variables or array elements which have been previously defined can be passed in the argument list.

Finally, the USR program must be correctly designed to receive argument addresses and control. In the MULT example, the routine should be coded:

```
        ORG     30000
CX      DEFW    0
AX      DEFW    0
BX      DEFW    0
          .
          .
          .
  -executable code-
          .
          .
          .
```

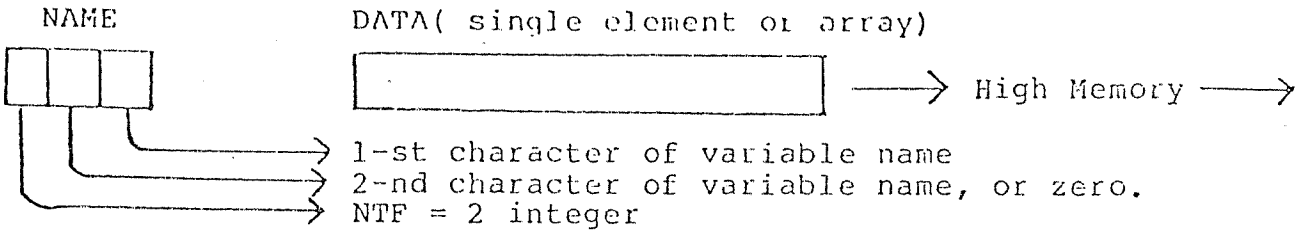
That is, control is not passed to location 30000. Rather, the address of each argument in the argument list is loaded beginning at 30000 and control is passed to the address after the last address load point. For an example of use, suppose C were an integer. To get the value of C in HL, use:

```
        LD      IX,(CX)
        LD      L,(IX)
        LD      H,(IX+1)
```

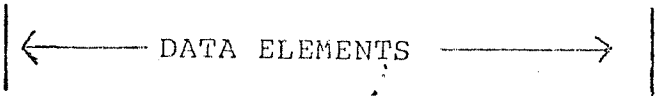
The advantages and disadvantages of passing argument addresses are:

1) By changing the value at the address passed, the USR program changes the value of the argument as seen by the BASIC program.

2) The receiving program can process any type of data by looking up the associated number type. (See Table 7 for data organization.) Passing values and number types would be an alternative. Another alternative would be to convert and pass numbers in double precision.



DATA:



ARRAYS

DIM X(5,7)

Byte	Integer	Single	Double	String	Format	Ex.(HEX)
0	LSB (1)	LSB(1)	LSB(1)	LENGTH(1)	LSB Size(2)	65
1	MSB	.	.	LSB String	MSB	00
2		MSB	.	MSB Adrs.	# Dimensions	02
3		EXP	.		LSB Value 1-st	08
4					MSB Dimension	00
5			.		.	
6			MSB		.	
7			EXP		LSB Value Last	06
					MSB Dimension	00

High Memory

. . . 1st Data Element (3),(1)

. . .

(1) Address returned by CALL 260DH.

(2) Size = # bytes used by # Dimensions, Dimension values and data elements.

(3) Data elements of the array are ordered by varying the first index the fastest. For example, the ordering is X(0,0), X(1,0), X(2,0) ... X(5,0), X(0,1), X(1,1), ..., X(5,7).

4) Passing values with number type conventions makes for the easiest loading in the user program.

5) Passing values allow arguments to be expressions. Passing addresses require simple variables or array elements, (array A is really array element A(0,0,...)).

There are several useful routines related to argument passing. These routines are summarized in Table 8 and facilitate the DEFUSR program and possible variations.

The variable location and creation logic at 260DH returns the address of the variable (ASCII string) pointed to by the HL register pair. After execution, HL points to the character following the last character of the variable name. If a variable cannot be found, it is created.

Be careful, variable creation may cause the dynamic relocation of other variables or arrays.

This will cause previously acquired variable addresses to be questionable. The variable address is returned in the DE register pair. For NTF = 2, 4 or 8, (integer, single, or double precision) the DE address returned is that given by the VARPTR logic in BASIC, (see Reference Manual 8/8-8/11 and Table 7). For NTF = 3, (strings), the DE address returned points to the first of three bytes which contain the string length and string address.

The variable to ACC and NTF logic located at 2540H returns the value of the variable (ASCII string) pointed to by the HL register pair. After execution, HL points to the character following the last character of the variable name. If a variable cannot be found, it is created and given a value of zero. The value is returned in the ACC and NTF. For NTF=3, (strings), the ACC contains the address of the first of three bytes which contain the string length and string address.

The expression to ACC and NTF logic located at 2337H returns the value of the expression, (ASCII string terminated with delimiters ")", ",", ":", or zero byte). On execution, the HL register pair points to the first character of the expression. After execution, HL points to the delimiter. In the case of string expressions, the ACC contains the address of the first of three bytes which contain the string length and string address. Note, CALL 2337H expects the machine to be formatted for BASIC RUN mode. Also, CALL 2337H makes significant use of the stack.

TABLE 8 USEFUL INTERFACE ROUTINES

1) Variable Location and Creation: CALL 260DH

Input — HL address of 1-st character.

Output — HL updated to character following last character of variable.

DE address of variable.

2) Variable --> ACC, NTF: CALL 2540H

Input — HL address of 1-st character.

Output — HL updated to character following last character of variable.

ACC,NTF value of variable.

3) Expression --> ACC, NTF: CALL 2337H

Input — HL address of 1-st character. String must terminate with appropriate delimiter; ")", zero byte , "," or ":" are satisfactory.

Output — HL address of delimiter.

ACC, NTF value of expression.

(See text for discussion)

A P P E N D I X A

MEMORY MAPS

Table 9 is a memory map of the Level II BASIC ROM. It has an entry point for each of the Level II keywords. Hence, keywords which have not been discussed in this document can be investigated with the help of a disassembler. In addition to keywords, entry points for other logic useful to assembler programming are included.

Table 10 lists the entry points for DISK BASIC keywords. It is useful to understand how DISK-BASIC is incorporated into Level II BASIC, (see the DEFUSR program in Section 7). Each DISK-BASIC command passes control to a unique address between 4152H-41A3H. If Level II BASIC is in use, these RAM addresses pass control to L3 ERROR (DISK BASIC only). If DISK BASIC is in use, control passes to the appropriate routine. Table 10 lists these link addresses. In addition to these link addresses, 41A6H - 41E2H link to DISK BASIC code to perform a number of maintenance functions or extensions of Level II commands. These include USR at 41A9H, error processing for extended messages at 41A6H, (can be used to trap errors - but the stack is lost), and program initialization at 41BBH.

TABLE 9 LEVEL II ROM MAP
(all addresses are in hexadecimal)

```
0008  RST 8;TEST (HL)-((SP)). IF NOT ZERO,SN ERROR. THEN RST 10H
0010  RST 10H:INC HL FIRST,WORK THRU STRING, IGNORE CR & SPACES. SET C IF NEXT CHARACTER
      A NUMBER, ELSE RESET C.
0018  RST 18H; Z, C, & S FIXED FROM INTEGER COMPARE HL-DE. MAINTAINS BC, DE, HL.
0020  RST 20H: IF NTF=8 C IS RESET ELSE C SET.  A=NTF-3 FIXES S & Z FLAGS.  MAINTAINS BC,
      DE, HL.
002B  KEYBOARD SCAN. RETURNS ASCII CHARACTER IN A. MAINTAINS BC, HL.
0033  DISPLAY BYTE IN A ON VIDEO. MAINTAINS BC, HL.
0049  GET CHARACTER FROM KEYBOARD & PUT IN A (WAITS FOR KEYSTROKE)
0060  DELAY LOOP. BC=68828*SECONDS. MAINTAINS D , HL.
0133* POINT. SEE NOTES TO USE GRAPHICS.
0135* SET
0138* RESET
019D* INKEY$
01C9* CLS
01D3* RANDOM
       ------
       TAPE LOGIC
       ------
01D9  WRITE WAVEFORM
01F8  TURN TAPE OFF
0212  DEFINE DRIVE
0235  READ BYTE
0241  READ BIT
0264  WRITE BYTE IN A TO TAPE
0287  WRITE LEADER
0296  LOOK FOR SYNC BYTE A5H.
02B2* SYSTEM
032A  DISPLAY BYTE IN A USING DEVICE TYPE (409CH)
0361  KEYBOARD TO STANDARD BUFFER & DISPLAY, (FOLLOW BY INC HL OR RST 10H).
03E3  (03E3 - 0457) ---------- KEYBOARD DRIVER
0458  (0458 - 058B) ---------- VIDEO DRIVER
05D9  KEYBOARD TO BUFFER AT HL & DISPLAY.  HL RETURNED UNCHANGED
070B  FADD: ACC=(HL) + ACC
0710  FSUB: ACC=(HL) - ACC
0713  FSUB: ACC=BCDE - ACC
0716  FADD: ACC=BCDE + ACC
0809* LOG: ACC=LOG(ACC). SINGLE IN AND OUT.
0847  FMULT: ACC=BCDE*ACC
08A2  FDIV:  ACC=BCDE/ACC
0955  CHECK ACC FOR ZERO
0977* ABS: ACC=ABS(ACC). INTEGER IN & OUT, OR SINGLE IN & OUT. NTF REQUIRED & MAINTAINED.
0982  ACC= -ACC: SINGLE ONLY. MAINTAINS BC, DE.
098A* SGN:  ACC & NTF INPUT FLOAT OR INTEGER.
            ACC=SGN(ACC)=-1,0,1 & NTF=02(ACC INTEGER).
0994  CHECK SIGN OF ACC, FLOAT OR INTEGER. REQUIRES NTF.  A=00 IF ACC=0. A=01 IF ACC
      GREATER THAN 0. A=FF IF ACC LESS THAN  0.  S & Z FLAGS ALSO REPRESENT ACC.
09A4  LOAD SINGLE ACC TO STACK. TO RETRIEVE
      POP   BC
      POP   DE.
      LOAD MAINTAINS A,BC, HL.
09B1  LOAD SINGLE:   ACC=(HL),(HL+1) ....  LEAVES HL=HL+4
```

09B4 LOAD SINGLE: ACC=BCDE. MAINTAINS HL.
09BF LOAD SINGLE: BCDE=ACC.
09C2 LOAD SINGLE: BCDE=(HL),(HL+1) .... LEAVES HL=HL+4

DATA MOVE ROUTINES

	BYTE COUNT	FROM ADDRESS	TO ADDRESS
09D7	B	DE	HL
09D6	A	DE	HL
09D3	NTF	DE	HL
09D2	NTF	HL	DE
09CE	4	DE	HL
09CB	4	ACC	HL

09F4 LOAD SINGLE OR DOUBLE(NEED NTF): ACC=DTEM
09FC LOAD SINGLE OR DOUBLE (NEED NTF): DTEM=ACC
0A0C SINGLE COMPARE: ACC-BCDE:FIXES S & Z FLAGS.
0A39 INTEGER COMPARE: HL-DE: FIXES S & Z FLAGS. MAINTAINS BC.
0A4F DOUBLE COMPARE: ACC-DTEM: FIXES S & Z FLAGS.
0A78 DOUBLE COMPARE: DTEM-ACC: FIXES S & Z FLAGS.
0A7F* CINT:SINGLE: ACC=CINT(ACC): LARGEST INTEGER NOT GREATER THAN THE ARG. INTEGER: IMMEDIATE RETURN.
0A9A INTEGER LOAD: ACC=HL: NTF=02: MAINTAINS BC,DE.
0AB1* CSNG: INTEGER: 1-ST CONVERT TO SINGLE.
SINGLE: NOTHING
DOUBLE: 4/5'S ROUND TO SINGLE.
0ACC CONVERT INTEGER ACC TO SINGLE ACC.
0ACF CONVERT INTEGER HL TO SINGLE ACC.
0ADB* CDBL: ACC(DOUBLE)=ACC(INTEGER OR SINGLE). REQUIRES NTF
0AF4 TEST NTF=3 (STRING). IF STRING, RETURN, ELSE, ERROR. MAINTAINS BC,DE & HL.
0B26* FIX: TRUNCATE FLOAT TO INTEGER & RETURN FLOAT. IF INTEGER, IMMEDIATE RETURN. IF STRING ERROR.
0B37* INT:RETURNS LARGEST SINGLE PRECISION WHOLE NUMBER NOT GREATER THAN ARGUMENT. INTEGER, IMMEDIATE RETURN. STRING, ERROR.
0BC7 SUB: ACC=HL=DE-HL: NTF=2
ON OVERFLOW DOES IN SINGLE & RETURNS SINGLE.
0BD2 ADD: SEE SUB.
0BF2 MULT: SEE SUB.
0C4C INTEGER ABS: ACC=HL=ABS(HL):NTF=2: MAINTAINS B,DE. SEE ABS AT 0977.
0C51 INTEGER CHANGE SIGN: ACC=HL=-HL:NTF=2: MAINTAINS C,DE.
0C6B INTEGER TO SINGLE: ACC(SINGLE)=DE(INTEGER): SEE 0ACF.
0C70 DSUB:(DOUBLE SUBTRACT): ACC=ACC-DTEM
0C77 DADD:(DOUBLE ADD): ACC=DTEM + ACC
0DA1 DMULT: ACC=ACC*DTEM
0DE5 DDIV: ACC=ACC/DTEM
0E65 ASCII CONSTANT TO ACC. RETURNS DOUBLE
0E6C ASCII CONSTANT TO ACC. RETURNS LEAST NECESSARY TYPE
0F8F POP THE STACK AND FADD.
0FBD ACC TO ASCII
13E7* SQR IN & OUT SINGLE ACC.
1439* EXP IN & OUT SINGLE ACC.
14C9* RND IN & OUT SINGLE ACC.
1541* COS IN & OUT SINGLE ACC. ANGLE IN RADIANS.
1547* SIN IN & OUT SINGLE ACC. ANGLE IN RADIANS.
15A8* TAN IN & OUT SINGLE ACC. ANGLE IN RADIANS.
15BD* ATAN IN & OUT SINGLE ACC. ANGLE IN RADIANS.

1608· TABLE OF BASIC FUNCTION ADDRESSES. ADDRESSES CORRESPOND TO THE SECOND HALF OF THE TABLE AT 165DH.
165D TABLE OF CHARS. MASK HIGH ORDER BIT AND USE TABLE AT 1822 TO DECODE FUNCTION NAME.
1822 2-ND TABLE OF BASIC FUNCTION ADDRESSES, ADDRESSES CORRESPOND TO THE FIRST HALF OF THE TABLE AT 165DH.

18C9 TABLE OF ERROR CODES ON PAGE B/1 OF LEVEL II BASIC MANUAL THIS TABLE CONTINUES TO 18F6.

191E FROM HERE TO 1936 IS A TABLE OF CHAR STRINGS. 'ERROR' 'IN' 'READY' 'BREAK'
1A19 RE-ENTER BASIC WITH READY.
1B49* NEW
1BB3 PROMPT & KEYBOARD INPUT TO STANDARD BUFFER.
1C90 COMPARE HL-DE UNSIGNED. FIXES Z FLAG ONLY.
1C96 RST 8 LOGIC. COMPARE (HL)-((SP))
1CA1* FOR
1D78 RST 10H LOGIC.
1D91* RESTORE
1DA9* STOP
1DAE* END
1DE4* CONT
1DF7* TRON
1DF8* TROFF
1E00* DEFSTR
1E03* DEFINT
1E06* DEFSNG
1E09* DEFDBL
1E7A* CLEAR
1EA3* RUN
1EB1* GOSUB
1EC2* GOTO
1EDE* RETURN
1F05* DATA
1F07* REM
1F21* LET
1F6C* ON
1FAF* RESUME
1FAF* ERROR
2008* AUTO
2039* IF
2067* LPRINT
206F* PRINT
219A* INPUT
21EF* READ
22B6* NEXT
2337 EVALUATE EXPRESSION, STORE IN ACC
2490 DIV: INTEGER DIVIDE: ACC=DE/HL: DE & HL ARE FIRST CONVERTED TO SINGLE. ACC RESULTS IN SINGLE.
24CF* ERR
24DD* ERL
24EB* VARPTR
2540 ASCII VAR TO ACC
25C4* NOT
25D9 RST 20H LOGIC: COMPARE NTF-3.
2608* DIM
260D ASCII VARIABLE LOCATION/CREATION. RETURNS ADDRESSES.

```
27C9* MEM
27FE* USR
27D4* FRE
27F5* POS
2836* STR$
28A7  DISPLAY A STRING
2A03* LEN
2A0F* ASC
2A1F* CHR$
2A2F* STRING$
2A61* LEFT$
2A91* RIGHT$
2A9A* MID$
2AC5* VAL
2AEF* INP
2AFB* OUT
2B29* LLIST
2B2E* LIST
2BC6* DELETE
2BF5* CSAVE
2C1F* CLOAD
2CAA* PEEK
2CB1* POKE
2E60* EDIT
```

* denotes Level II BASIC keywords.

TABLE 10 DISK BASIC ENTRY POINTS
(addresses in Hexadecimal)

CODE ADDRESS	COMMAND	LEVEL II LINK ADDRESS
558E	FN	4155
5655	DEF	415B
56C4	CMD	4173
5714	TIME$	4176
5756	LINE	41A3
582F	INSTR	419D
58B7	&	4194
5E2D	MK1$	416A
5E30	MKS$	416D
5E33	MKD$	4170
5E46	CVI	4152
5E49	CVS	4158
5E4C	CVD	415E
5F7B	LOAD	4188
600E	MERGE	418B
6044	SAVE	41A0
606F	CLOSE	4185
60AB	FIELD	417C
60E5	RSET	419A
60E6	LSET	4197
61EB	EOF	4161
6231	LOC	4164
6242	LOF	4167
627B	PUT	4182
627C	GET	417F
6346	NAME	418E
6349	OPEN	4179
63C0	KILL	4191

A P P E N D I X B

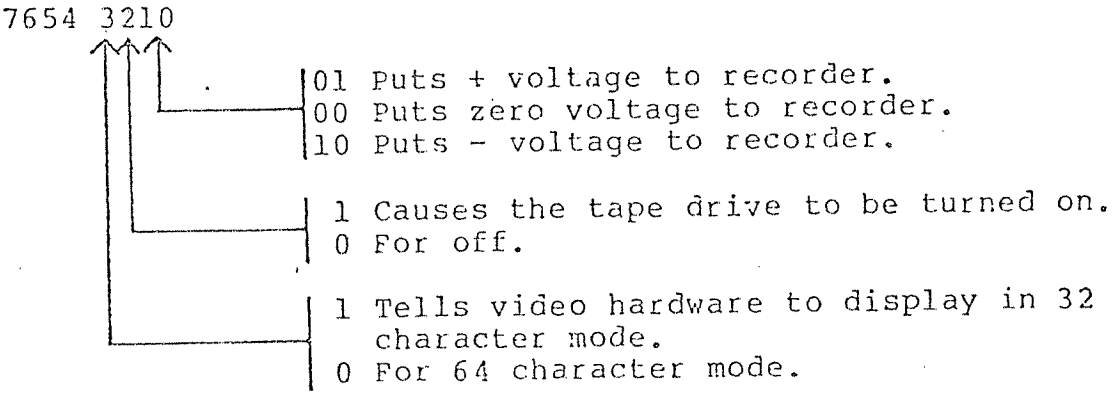
TAPE I/O AND SQUARE WAVE MUSIC

The output port number 255 controls the 32 and 64 character display modes, the tape on/off function and the output tape voltage. The contents of address 37E4H controls cassette #-1. Address 37E4H is defined by CALL 0212H, (see Section 6).

When...

```
OUT     (255), A
```

is executed, the format of A is:



Creating square wave music on the tape output line is a matter of outputting the correct number to port 255 at the correct time. An example of OUT instructions and timing loops is given in Example 9. This routine processes data pointed to by the IX register. First, DE is loaded with the number of square wave cycles to be in the note. Plus (+) and minus (-) voltages are sustained by the timing logic at 0060H, (0060H uses the A and BC registers only). The duration of timing delay is:

$$T(\text{in seconds}) = (1.453 * 10 **(-5)) * BC$$

The frequency (in cycles per second) if the note is given by

$$F=1/(Tp + Tm)$$

where Tp and Tm are durations (in seconds) of the plus and minus wave forms respectively. The duration of a note (in seconds) is given by:

$$d = (Tp + Tm) * DE$$

Finally, if N represents notes on the chromatic scale, with N=0 corresponding to middle C, then their frequency is given by:

$$F =EXP (.05776 N + 5.567)$$

EXAMPLE 9 Square Wave Note Routine

```
10          LD      E,(IX)          ; get number cycles
20          LD      D,(IX+1)
30 DURA     LD      A,5             ; + waveform
40          OUT     (255),A
50          LD      C,(IX+2)        ; set up delay loop
60          LD      B,(IX+3)
70          CALL    0060H           ; call delay loop
80          LD      A,6             ; - waveform
90          OUT     (255),A
100         LD      C,(IX+4)        ; set up delay loop
110         LD      B,(IX+5)
120         CALL    0060H           ;call delay loop
130         DEC     DE              ; decrement duration
                                      counter
140         LD      A,D             ; test duration
150         OR      E
160         JR      NZ,DURA         ; go repeat waveform
```

To create music, one generally wants a certain note, N, and duration, d. Solving for the above equations on the previous page, then:

BCp + BCm = EXP (-.05776N - 5.567)/1.453 * 10 ** (-5)

DE = d*EXP (.05776N + 5.567)

By splitting the total count BCp + BCm (where BCp is the value of the BC register delay count for the plus waveform and BCm is for the negative waveform, respectively pointed to by IX + 2 and IX + 4), notes having different tone result. Even splitting produces pure tones while uneven splitting produces more resonant tones.

A P P E N D I X C

TAPE AND BASIC MEMORY FORMATS

These formats are useful to know when you only have one copy of a tape that will not load. Example 6 (see Section 6) lists a program that would read such a tape into memory, where it can be edited. The program will also write the data out to a new tape. None of the formats discussed below include the 256 zero byte leader and A5H sync byte written by CALL 0287H.

EDTASM SOURCE TAPE FORMAT (from W command):

	D3	header
	54 45 53 54 41 41	file name "TESTAA" up to 6 characters.
(1)	B0 B0 B0 B1 B0	line # 10
	20	data header(character blank)
	XX XX XX XX . . .	data (09 as tab)
	0D	data trailer (character CR)
	.	
	.	
	.	
	1A	end of file mark

(1) Repeated record.

EDTASM OBJECT TAPE FORMATS:

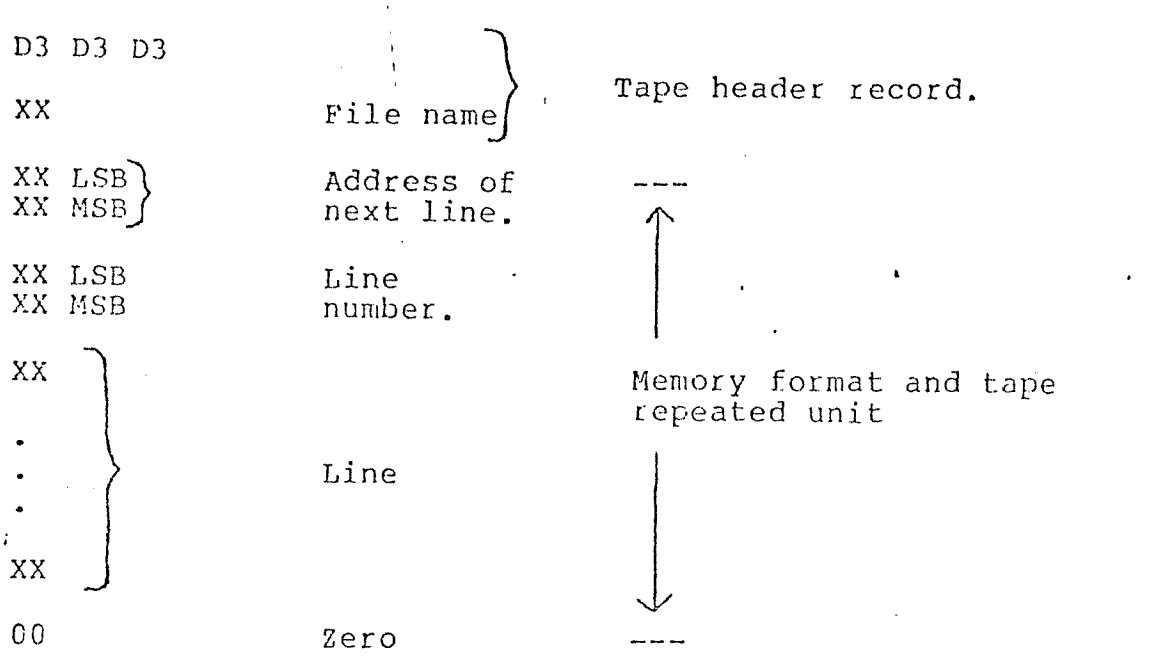
	55	header
	54 41 50 45 49 4F	file name "TAPEIO"
(1)	3C	Code (78 implied end of file)
	44	length of data portion (00 is a full record of 256 bytes)
	90 47	points to address of data load (LSB,MSB)
	XX XX XX XX . . .	data bytes
	23	check sum (add together data and load point)
	.	
	.	
	.	
	78	end of file mark
	6D 43	execution address (LSB,MSB)

(1) Repeated record.

. . .	character string	repeated unit.
2C	ASCII comma	

When a character string is output it goes into the above character string field unmodified. Variable outputs are preceeded and followed by a blank. For example, PRINT #-1, EXP(1) gives a character string field: b2.71828b.

BASIC TAPE AND RAM FORMAT:



For example, when CSAVE"F" is performed on the program

```
10  BB
20  CCD
30  DDDD
```

tape and memory appear

D3 D3 D3 46 Tape only, 46H = "F".

Address	Memory & Tape Contents	Comment
42E9	F0 42	
	0A 00	Line no. 10
	42 42	"BB"
	00	
42F0	F8 42	
	14 00	Line no. 20
	43 43 44	"CCD"
	00	
42F8	01 43	
	1E 00	Line no. 30
	44 44 44 44	"DDDD"
	00	
4301	00 00	

A P P E N D I X D

A FUNCTION TEST PROGRAM

The test program listed on the following page, serves two purposes. First, it allows many of the functions described in this manual to be verified by direct means. Second, it demonstrates (and implicitly tests) data conversion routines, display routines and data move routines. Except for initialization,, the routine returns to ..RET after CALLing a function. At that point, all registers, ACC, NTF and DTEM are saved because they will be affected by the data conversion routines. Next, the NTF and ACC are displayed. The user is then prompted for the (decimal) address of the test function. This is converted and stored at GOTO. The ACC, NTF and DTEM are then restored. In normal programming, a dynamic address, (the function address at GOTO) would be transferred by JP (HL). But, in this case, that would disturb the HL register pair. Instead, the return address, ARET, is PUSH'd, then (GOTO) is PUSH'd. The registers are restored and a CALL is achieved by executing a RET.

EXAMPLE 10 A FUNCTION TEST PROGRAM

```
00100 START   EQU     7000H
00200         ORG     START
00300         LD      SP,START-2
00400         LD      HL,1600H
00500         CALL    09B1H    ;ACC GETS -1/3
00600         LD      HL,40AFH
00700         LD      (HL),4
00800 ARET    EXX     ;SAVE GOOD RECS
00900         EX      AF,AF'
00902         LD      DE,412ZH          ;FROM ADDRESS
00904         LD      HL,TDTEM          ;TO ADDRESS
00906         LD      B,8               ;BYTE COUNT
00908         CALL    09D7H    ;SAVE DTEM
01000         LD      DE,411DH          ;FROM ADDRESS
01100         LD      HL,TACC           ;TO ADDRESS
01200         LD      B,8               ;BYTE COUNT
01300         CALL    09D7H    ;SAVE ACC
01400         LD      A,(40AFH)
01500         LD      (TNTF),A ;SAVE NTF
01502         ADD     A,48
01505         CALL    032AH    ;DISPLAY NTF
01510         LD      A,20H
01515         CALL    032AH    ;DISPLAY BLANK
01600         CALL    0F8DH    ;ACC TO ASC II
01700         CALL    28A7H    ;DISPLAY ASC II
01800 ;INPUT ADDRESS FOR EXECUTION.
01900         CALL    1BB3H    ;PROMPT & KEYBOARD TO BUFFER
02000         RST     10H
02100         CALL    0F4CH             ;ASC II TO ACC
02200         CALL    0A7FH             ;CINT
02300         LD      (GOTO),HL ;SAVE JUMP ADDRESS
02400         LD      DE,TACC           ;FROM ADDRESS
02500         LD      HL,411DH          ;TO ADDRESS
02600         LD      B,8               ;BYTE COUNT
02700         CALL    09D7H    ;RESTORE ACC
02800         LD      A,(TNTF)
02900         LD      (40AFH),A         ;RESTORE NTF
02905         LD      DE,TDTEM          ;FROM ADDRESS
02910         LD      HL,412ZH          ;TO ADDRESS
02915         LD      B,8               ;BYTE COUNT
02920         CALL    09D7H    ;RESTORE DTEM
03000         LD      HL,ARET
03100         PUSH    HL       ;PUSH RETURN ADDRESS
03200         LD      HL,(GOTO)
03300         PUSH    HL       ;PUSH CALL ADDRESS
03400         EX      AF,AF'
03500         EXX
03600         RET              ;MAKE CALL
03700 TACC    DEFS    8
03800 TNTF    DEFS    1
03900 GOTO    DEFS    2
03Z05 TDTEM   DEFS    8
04000         END     START
```