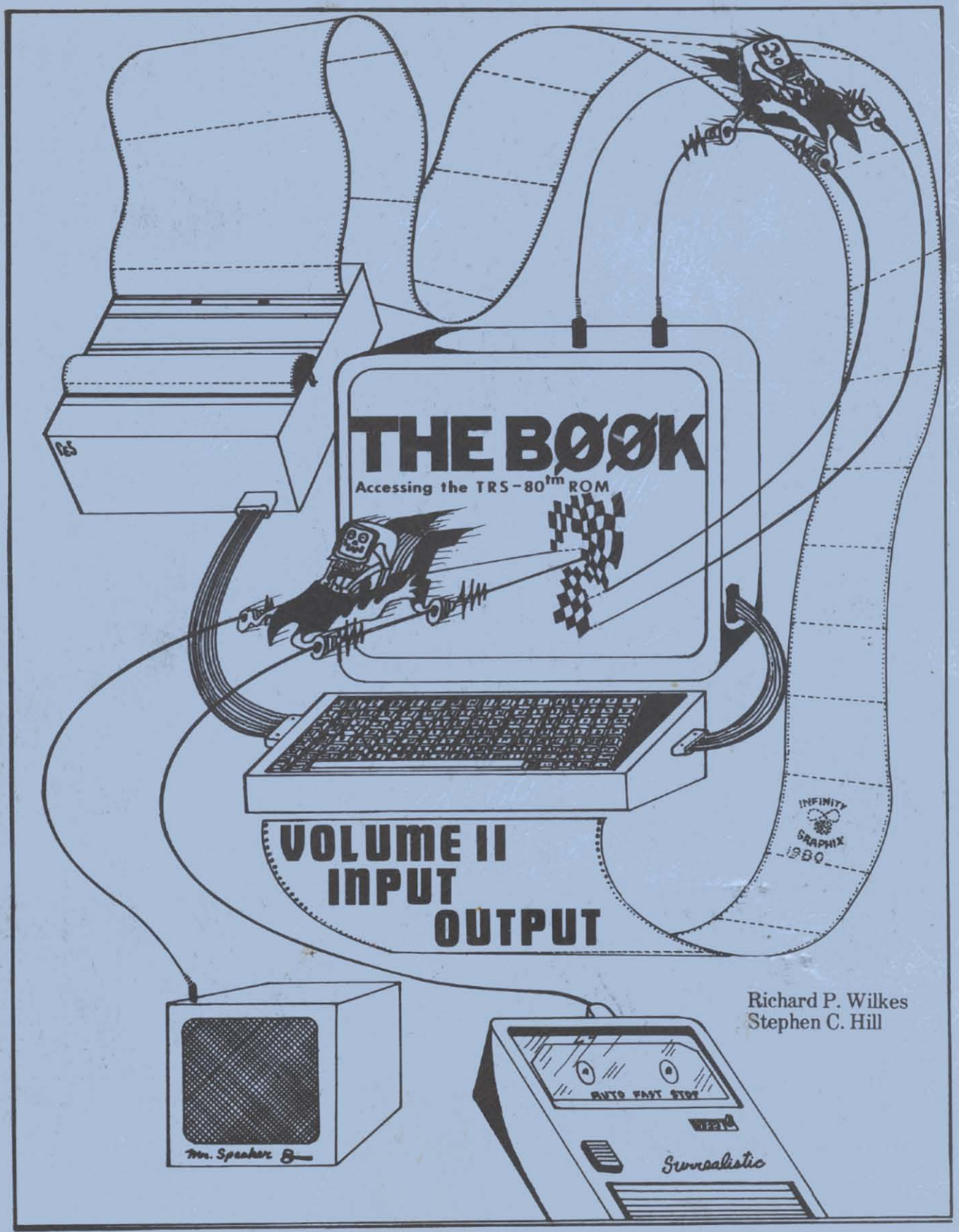
THE BØØK

VOLUME II

ISC



Richard P. Wilkes
Stephen C. Hill

THE BØØK

ACCESSING THE TRS-80 ROM

VOLUME II: INPUT/OUTPUT

Richard P. Wilkes
Stephen C. Hill

Technical Assistance
Roy Soltoff
Raymond E. Daly IV
Thomas B. Stibolt, Jr.

ILLUSTRATIONS BY
INFINITY GRAPHIX

Insiders Software Consultants
P.O. Box 2441
Springfield, VA 22152

Copyright © 1981 by Insiders Software Consultants, Inc.
All rights reserved.

First Edition 1981

Reproduction in any manner, electronic, mechanical, magnetic, optical, chemical, manual or otherwise, without expressed written permission, is prohibited.

Disclaimer:

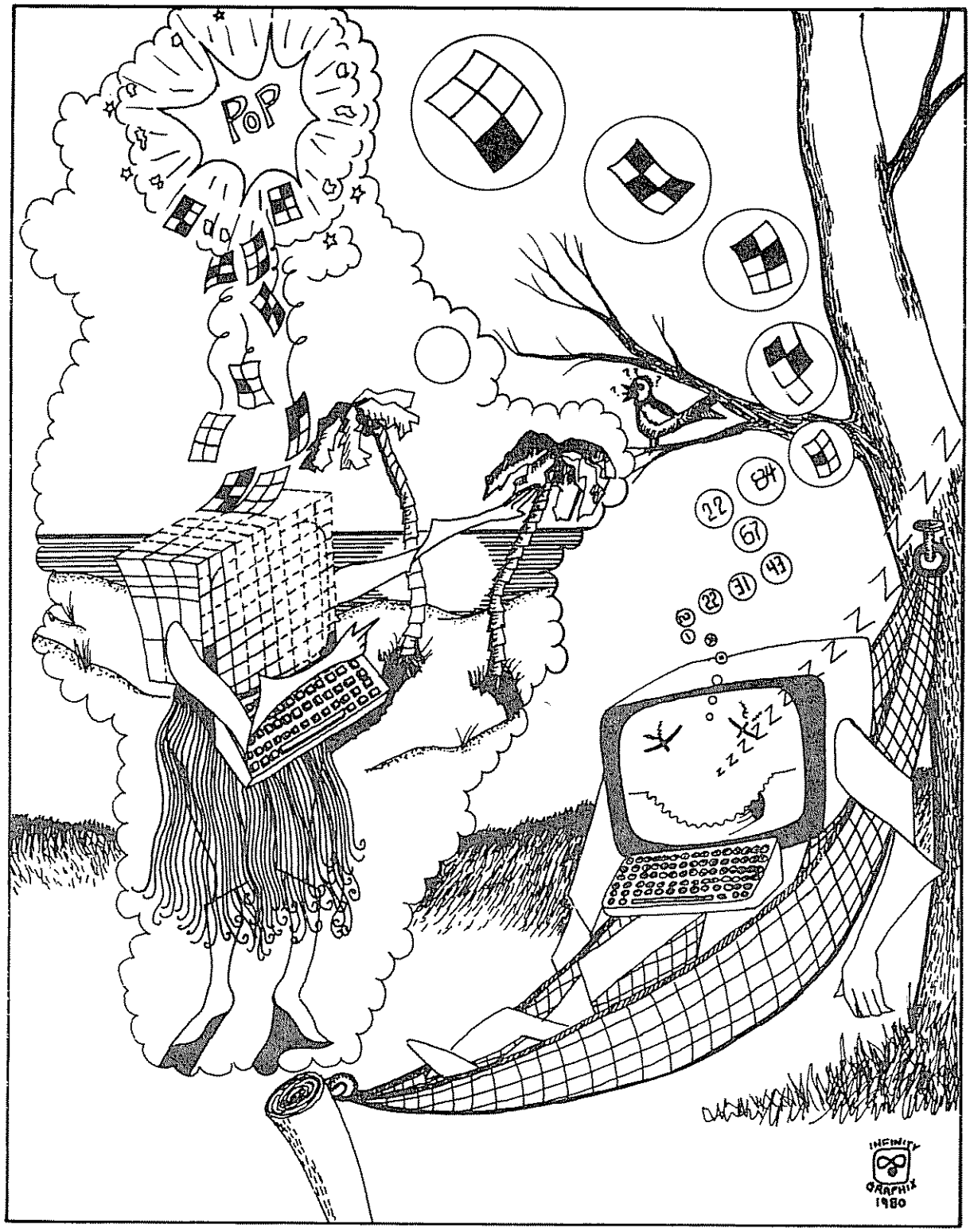
While Insiders Software Consultants, Inc. has taken every precaution in the preparation of this book, it assumes no responsibility for errors or omissions. Neither is any liability assumed for damages resulting from the use of the information contained herein.

Radio Shack™ and TRS-80™ are registered trademarks of the Tandy Corporation.

Insiders Software Consultants, Inc.
PO Box 2441,
Springfield, VA 22152

To Mom, Sharon, Mr. Irwin, and Steve
without whose love and encouragement through
many trying times I could never have made it.
-RPW

I would like to dedicate my portion of this book
to my Mother, who is still amazed; my wife Sheila,
who took charge of the operations of the company
allowing us to finish this volume; and to
Doug Manley, who showed me the Magic.
-SCH



GRAPIX'S MODE for the "Book Vol. II

INFINITY GRAPHIX 1980

Table of Contents

Table of Contents

Table of Contents

Preface

Information in this book is presented in sequential order. The basic concepts presented in Chapter One are used as the foundation for the more concise information found in the later sections. You should become comfortable with the introductory chapter before moving to the other specific discussions of I/O. After Chapter One, the other sections may be perused in any order, but each individual chapter should be read completely. Do not skip directly to the interfacing examples at the end of the chapter.

An alphabetical index is not supplied with this volume. As was done with Volume I, the detailed Table of Contents should be used to locate quickly items of interest.

This book was written using the Scripsit word processing package from Radio Shack (Catalog 26-1563) with the SuperScript modification from Acorn Software Products; 634 North Carolina Avenue, SE; Washington, DC 20003. Assembled programs found in the appendices were written under the disk-based editor/assembler EDAS 3.4 from MISOSYS; 5904 Edgehill Drive; Alexandria, VA 22303. The text was printed using an NEC Spinwriter Model 5530.

Introduction

Ah, the well laid plans of mice and men... that's what comes to mind when we think of this volume. It was supposed to appear in September 1980, then October, November, and December. Finally, in the first days of January 1981, all the technical difficulties have been solved, and we have Volume II of **The BØØK: Accessing The TRS-80 ROM.** To those of you that have patiently awaited this volume, thank you. We think you will be pleased with the results.

Several books have appeared on the market which cover "completely" the workings of the TRS-80 ROM. It seems strange that anyone could completely cover a 12K program in less than 200 pages. When interfacing to ROM routines, one has to know the full story, not simply a brief explanation. We have tried to take the guesswork out of interfacing. For this reason, we have devoted an entire volume to the math routines and an entire volume to input/output.

Provided in this volume is a complete interfacing guide for each of the Level II I/O units. You will note that we have spent a great deal of space in each chapter explaining the HOW's and WHY's of each device. It is easy to use just the ROM routines, but as you write more sophisticated software, you will probably have to control the devices directly. The technical information given in each chapter is vital when writing customized drivers.

And, of course, we all can learn from analyzing another person's code. If you have a TRS-80 or are writing programs for one in assembly language, you have probably disassembled the ROM; that's where it starts. Unfortunately, the process of commenting bare, disassembled code takes many painstaking hours. Take a look in chapter 6; in it you will find the comments for the I/O routines--all you need are the operands (space is provided for you to enter them).

In the appendices, you will find useful lists, tables, and assembly language routines, including an alphabetized list of the labels found in Appendix A of Volume I, a lowercase driver, a table of graphics characters, a SET/RESET/POINT routine, and an ASCII table.

Completeness and accuracy have been the major goals in this series. We welcome any constructive comments, additions, or suggestions. Please feel free to write us.

Thank you and happy computing,

Insiders Software Consultants, Inc.
PO Box 2441
Springfield, VA 22152
U.S.A.

Introduction to Input/Output on the TRS-80

This volume is dedicated to two very important aspects of every computer--the **input** and **output** (I/O) of data and information in a form that can be easily understood and manipulated by both the computer and you, the owner, operator, and user of the TRS-80. Throughout this volume, the term "TRS-80" only refers to the TRS-80 Model I microcomputer sold by Radio Shack, a division of Tandy Corporation. As mentioned in the Introduction, descriptions and discussions will be kept as simple as possible. However, some knowledge of Z-80 assembly language is helpful.

There are two major methods used to input information into the TRS-80; the primary source is the keyboard. A full discussion of the keyboard and the **Read Only Memory** (ROM) routines used in conjuction with this device are provided in chapter 2. The second device used for input is the cassette tape unit, discussed in chapter 5.

Although there are other means that may be used to input data into the computer, these are the only two that are supported by the Level II ROM. One must remember that the information provided applies ONLY to Level II. When programming under a **Disk Operating System** (DOS), it may be necessary to add certain precautionary steps to any assembly language programs which use ROM routines for I/O. Whenever possible, problem areas are noted, but the ultimate responsibility lies with the programmer.

There are three devices which are used for output. They are the video display or **Cathode Ray Tube** (CRT), the hardcopy device or PRINTER, and the cassette TAPE. Of these units, the video display is used most often. It is discussed in chapter 3. A printer may be attached to the TRS-80 Level II system to produce hardcopy, although a special cable is necessary. Information on the printer routines is provided in chapter 4. The use of the cassette tape as an output device is included in chapter 5.

The Level II ROM uses a well defined procedure to access, or drive, most of the devices. The procedure consists of six steps. The first step is the definition of the data flow [either user-to-computer (input) or computer-to-user (output)]. The second step is the selection of a **Device Control Block** (DCB) by identifying its address in RAM memory. The third step consists of saving most of the CPU registers in order to preserve their contents during the I/O driver execution. Then, the driver routine is executed. The step after execution is the restoration of the registers. Finally, control returns to the calling procedure, with the registers in various states depending on the I/O request.

Each of the above steps will be discussed at length in the following sections. It is recommended that you become comfortable with each section before continuing. A complete understanding of these points is assumed in later chapters.

For clarity, only the keyboard, video, and printer devices will be used as examples. The tape unit does not conform to the above procedure; consequently, it is left out of this introductory discussion to avoid confusion. The method of utilizing the tape unit, as noted previously, is fully discussed in chapter 5.

DATA FLOW

Depending on the device, data may flow TO the device, FROM the device, or TO and FROM the device. The ROM I/O routines provide a method of avoiding possible conflicts in I/O requests by passing a flag byte from the calling procedure to the I/O routine. This byte is compared to the FLAG byte stored in the DCB. If a VALID request is made, execution continues. However, if an INVALID request is made, such as output to the keyboard, a routine which handles invalid requests is executed.

The determination of the validity of the request is performed by the I/O master driver found in ROM at location 03C2H, hereafter referred to as **DRIVER**. The DRIVER routine is usually accessed by a CALL to either the **INBYT** routine located at 0013H or the **OUTBYT** routine at location 001BH. Before the call, the DE register pair (referred to as 'DE'), is loaded with the address of the DCB.

These two routines (INBYT and OUTBYT) simply define the data flow direction. INBYT, which loads the flag byte in register B ('B') with 01H, informs DRIVER that the flow will be to the calling routine from the device. OUTBYT loads 'B' with 02H which denotes an output operation; data flow is from the calling routine to the device.

Another routine is present in the ROM for sending a control byte to a device. This CALL (to location 0023H) is not used by the ROM, but is present for other user-defined drivers. **CTLBYT** loads 'B' with 04H and then passes control to DRIVER for processing of the request.

As the reader may have noticed, the flag bytes placed in the 'B' register only have one bit set. The flags are used as follows:

```
Bit 0:  0000 0001 = 01H     ;INPUT   flag
Bit 1:  0000 0010 = 02H     ;OUTPUT  flag
Bit 2:  0000 0100 = 04H     ;CONTROL flag
```

Now, let us look at the DCB's for the different devices. As previously mentioned, the DRIVER checks the DCB for the flag byte. This flag byte is stored in the first byte of the DCB. The initial values are as follows:

```
4015:     KBTYP   DEFB  01H   ;KEYBOARD: Input only
401D:     CRTTYP  DEFB  07H   ;CRT:      In/Out/Control
4025:     LPTTYP  DEFB  06H   ;PRINTER:  Out/Control
```

Careful examination of these values shows that a given device may be able to handle one, two, or all three types of requests. A set bit (Bit is high, or 1) denotes that the device addressed by this DCB is capable of the corresponding request. For example, if the request is for input, one can see that both the keyboard and video are capable of input, but the printer is not. A quick note: the use of the video display for input is not user input but input from the screen memory. This may not make sense here, but it will become more clear after the discussion of the video in chapter 3. For now, just remember that it is capable of input.

DCB DEFINITION

The Device Control Block contains information which is dependent on the type of device it describes. However, for Level II devices, the first three bytes are always used for the same purpose. The first byte was just discussed and is the I/O request flag used in data flow. The next two bytes contain the address of the device driver, which contains the I/O unit dependent code. In other words, a keyboard and a video display require different "programs" to make them work in an acceptable manner. Therefore, after a certain amount of validity checking, the master driver must pass control to a routine which handles the I/O device.

As is the case with most addresses stored in memory, the location of the driver is stored with the **Least Significant Byte** (LSB) first, followed by the **Most Significant Byte** (MSB). When DRIVER gets control, the address of the DCB is (or atleast should be) in 'DE'. DRIVER then loads this into 'IX' and uses this index register to pick out the device driver address and load it into 'HL'. Here is an example of a user-defined DCB located at 7000H which is used to define an input device driver at address 735AH:

```
7000:       01H    (IX + 00H)    ;Type flag = INPUT
7001:       5AH    (IX + 01H)    ;LSB Driver address
7002:       73H    (IX + 02H)    ;MSB Driver address
7003:       .      (IX + 03H)    ;Misc status bytes
  .         .         .                 .
  .         .         .                 .
7007:       .      (IX + 07H)           .
```

DRIVER takes this DCB and loads the LSB into 'L' using the instruction "LD L,(IX+01H)" and then loads the MSB into 'H' using "LD H,(IX+02H)." Transfer is then passed to the I/O handler for the device using a "JP (HL)" instruction. The return address to DRIVER, DRVRET, has been placed on the top of the stack by a "PUSH HL" at 03CCH. Therefore, a RET instruction in the handler will return to DRIVER at the point in which it restores the registers. Control then passes to the CALLing routine by RETurning to the address at the top of the stack.

REGISTER USAGE

Obviously, during the execution of the I/O process some registers are going to be used. DRIVER saves almost all registers. Here are some important points to remember:

1) Level II does not use the alternate register set; therefore, DRIVER does not save these registers. If a user defined driver utilizes these registers, be certain that it does not destroy their values.

2) All current-set registers except 'DE' & 'AF' are stacked before execution of the I/O process. The stack during the I/O handler execution is as follows:

```
TOP OF STACK: DRVRET        ;Return Address to
                            ; DRIVER to restore
                            ; registers that
                            ; follow.
              'DE'          ;DCB address
              'IX'          ;IX before I/O Req.
              'HL'          ;HL before I/O Req.
              'BC'          ;BC before I/O Req.
                            ; BC is PUSHED by
                            ; INBYT/OUTBYT/CTLBYT
              RETADR        ;Return address to
                            ; CALLing routine.
```

3) The 'IY' register is not used at all by Level II. If it is not used in the driver, there is no need to save it. This fact contradicts the Radio Shack assembler manual.

4) DRIVER sets up the registers and flags as follows before transfering control to the device driver:
   a) Stack as shown above
   b) 'C' contains the old contents of 'A'
   c) 'HL' contains address of driver
   d) 'IX' contains address of DCB
   e) The Carry flag is set if Input request
   f) The Zero flag is set if Output request
   g) Test for No Carry for Control-byte request

Careful consideration of the above points is necessary for proper utilization of the ROM routines. You must be certain that the environment and stack are preserved at all times. Failure to do so may cause unpredictable results or the crashing of the computer (in other words, the program may wind up in OZ when it was looking for Kansas). The structure of the DCB allows for a great deal of flexibility; yet, much caution must be taken when manipulating these areas to avoid system errors.

DRIVER ROUTINES

The driver routines are as varied as the devices they control. The video routine must perform the scrolling, cursor positioning, manipulation of the screen depending on the control character sent, and many other sophisticated functions. The keyboard routine must "scan" the keyboard memory (a concept that will be discussed later) to determine if a new character has been pressed and then decode the scan into the appropriate character. Other drivers may be equally as complicated.

At some time in the future, you may wish to replace a ROM driver with another, more customized driver. Of course, some compatibility must remain in order for the machine to function properly. It is recommended that the programmer of such a driver analyze the ROM drivers first, paying close attention to the one that is to be replaced. After careful study, design the driver to suit the requirements of the situation, but be very cautious. The programmer must take into consideration many different, varying environments. The ROM is a complicated world.

At this stage, an example may be necessary to prove the point. Let's take the video driver. The video driver does a lot of work! It tracks cursor position on the screen, position in the line, whether the cursor is on or not, whether the cursor is hiding a character, and whether the screen is in 32-character mode, which changes just about everything. Once again, be very thorough; it pays off in the end.

Each of the drivers is discussed in their respective chapters, so very little needs to be added at this point. The complexity of a replacement driver is only limited by creativity. For instance, one could develop a sophisticated video driver with blinking cursor, direct cursor addressing, multiple pages, right-to-left and left-to-right scrolling, variable tab stops, upper and lowercase, etc. True, programming these features may be beyond one's patience or capability. Nevertheless, they are possible.

When you load a new driver into RAM, the items that you need to change are the address of the driver in the DCB and the flag byte. Also, other storage areas in the DCB must be changed to suit the new driver. Be sure that the driver is fully operational before modifying the DCB to avoid crashing the device and possibly the system.

REGISTER RESTORATION AND I/O COMPLETION

After the device driver is finished, control is passed to DRIVER, which restores the registers (except 'DE' and 'AF') and control is returned to the calling routine. If 'DE' is to be restored to its original state, it must have been saved by the caller before the request.

All flags and values from the ROM drivers are passed in 'AF'. After testing the flags or storing the returned data (if any is returned), 'AF' can be restored IF it was saved earlier along with 'DE'.

This completes the I/O driver procedure. Some of the points made will be clarified and expanded in the individual chapters pertaining to the devices. At this point, if the reader is unsure about the procedure, the above sections should be re-read.

INVALID REQUESTS

A special note to programmers:

When an invalid request is made (such as input from the printer), the ROM passes control to 4033H. This is the first address of a three-byte area used to store a JP to a handling routine. However, under normal Level II, these addresses contain:

```
4033:   LD    A,0          ;Clear Accumulator
4035:   RET                ;Back to DRIVER
                           ; Restore registers
```

This short routine simply clears 'A' and returns to DRVRET to restore the registers and return to the caller without executing a device handler.

It should be noted that almost ALL disk operating systems place a vector at these locations which causes execution of a disk I/O driver if an invalid request is made; results are unpredictable and could be fatal. Consequently, if one is going to experiment with the DCB's, be sure to control these addresses.

Input: The Keyboard

The primary device used for input to the TRS-80 is the keyboard; it is also perhaps the hardest device to understand. Many users think that when one presses the "A" key that the keyboard sends the character "A" to the computer. This is not the case as we shall soon see.

The TRS-80 keyboard consists of 53 single-pole, single-throw, normally open keys (65 with the numeric keypad). Each of the keys is assigned a location in a keyboard matrix. When a switch is closed (key pressed), it will short out a horizontal line to a vertical line, causing the appropriate element in the array to be "activated." In other words, when a key is pressed the contact causes an electrical "short." [This does not mean that your keyboard will burst into flames if you type too quickly. In this case, electical short simply means an electrical "connection".] This short is translated into a "1" bit in the assigned location in the matrix. This bit will stay on until the key is released.

What does this all mean? Well, it means that the task of figuring out the ASCII value of the key pressed on the board is not as easy as one might hope. First of all, the only thing we have to work with is a matrix of signals. In addition, more than one signal may be on at a time. Also, a signal stays on as long as the key is pressed. Therefore, there must be a method of decoding and then a method of determining whether the key was pressed during the last scan in order to make sense out of the matrix. Before we can intelligently discuss the matrix and decoding, we must define the concept of memory mapping. After the introductory sections, we will present the actual interfacing to the ROM routines.

MEMORY MAPPING

There are two techniques used to access I/O devices in the TRS-80. One method uses the Z80 I/O ports. The devices are accessed by performing IN's and OUT's to these ports; the tape unit uses this method. On the other hand, the method used by both the keyboard and the video is **memory mapping.** Each device is accessed by referring to its addresses in memory. Let's take an example. We'll use the printer since it is the easiest to understand.

The printer uses the address 37E8H. Meaning, if you want to read the status of the printer, you would perform an LD A,(37E8H). This instruction would retrieve the status of the printer into the accumulator. After testing the status to see whether the printer is ready, we could send an ASCII character stored in 'A' to the device by performing a LD (37E8H),A.

This example shows that when you specify the address 37E8H in an assembly language program on the TRS-80, you are not accessing a ROM or RAM address but a device. The specifics regarding the printer are discussed in a later chapter. What we are trying to stress here is that some devices on the TRS-80 are operated using instructions which normally refer only to memory. Therefore, if you store a byte in an address used by a device, and you read it back from the address later, it may not be the same. For this reason, you should not use the device addresses as storage areas.

The keyboard and video are addressed similarly. All three devices use addresses between the end of the ROM at 2FFFH and the beginning of RAM at 4000H.

Memory mapping is used for the keyboard, video, and printer. When a device is said to be "addressed" at a certain hex location, it means that the I/O device is memory mapped to that location. Keep this fact in mind in the following sections.

DECODING AND THE MATRIX

The keyboard matrix (KEYMEM) is addressed at 3800H and continues to 3BFFH. The matrix is actually made up of eight "rows" and eight "columns," where the addresses make up the rows and the values that are read from the addresses make up the columns. For the time being, we will ignore the MSB of the address and concentrate on the low order byte.

When scanning the keyboard, the following addresses are used: 3801H, 3802H, 3804H, 3808H, 3810H, 3820H, 3840H, and 3880H. The binary representation of the LSB's are as follows:

Bit #:	7	6	5	4	3	2	1	0
01H =	0	0	0	0	0	0	0	1
02H =	0	0	0	0	0	0	1	0
04H =	0	0	0	0	0	1	0	0
08H =	0	0	0	0	1	0	0	0
10H =	0	0	0	1	0	0	0	0
20H =	0	0	1	0	0	0	0	0
40H =	0	1	0	0	0	0	0	0
80H =	1	0	0	0	0	0	0	0

You should note that only one bit is set and that bit may be located in one of eight different locations. In this manner, the TRS-80 defines one of eight "rows."

In each row, we can have up to eight values. Let's assign each of these values to an individual bit and look at the matrix.

Column bit:		7	6	5	4	3	2	1	0
		80H	40H	20H	10H	08H	04H	02H	01H
Row bit: 7	[80H]				CTL				SFT
6	[40H]	SPA	R.A	L.A	D.A	U.A	BRK	CLR	ENT
5	[20H]	/	.	-	,	;	:	9	8
	(w/SFT)	?	>	=	<	+	>	)	(
4	[10H]	7	6	5	4	3	2	1	0
	(w/SFT)	'	&	%	$	#	"	!	
3	[08H]						Z	Y	X
2	[04H]	W	V	U	T	S	R	Q	P
1	[02H]	O	N	M	L	K	J	I	H
0	[01H]	G	F	E	D	C	B	A	@

ENT = Enter CLR = Clear BRK = Break
U.A = Up arrow D.A = Down arrow R.A = Right arrow
L.A = Left arrow SPA = Space SFT = Shift
CTL = Electric Pencil Control

At this point, an example would help to pull together this matrix concept. Let's look at row 1. It is assigned to bit 1 in the address table. We note that the LSB for this row is 02H (bit 1 = 02H); therefore, we will peek at location 3802H to look at this row.

```
LD      A,(3802H)
```

We find the value in 'A' to be 04H. Looking at the above table, we see that this corresponds to the letter 'J' which is at bit 2. In this manner, the TRS-80 locates the correct column by the value read from the row address.

That is pretty straight forward, but what happens when more than one key is pressed in the same "row?" For example, what would be the value at 3802H if both the letter 'J' and the letter 'M' were pressed. Simply, the letter 'J' would cause bit 2 to be set, and the letter 'M' would cause bit 5 to be set. Then, adding (or in assembly langauge, ORing) the appropriate values for these bits, we get 04H + 20H = 24H, which is the value at 3802H when 'J' and 'M' are pressed.

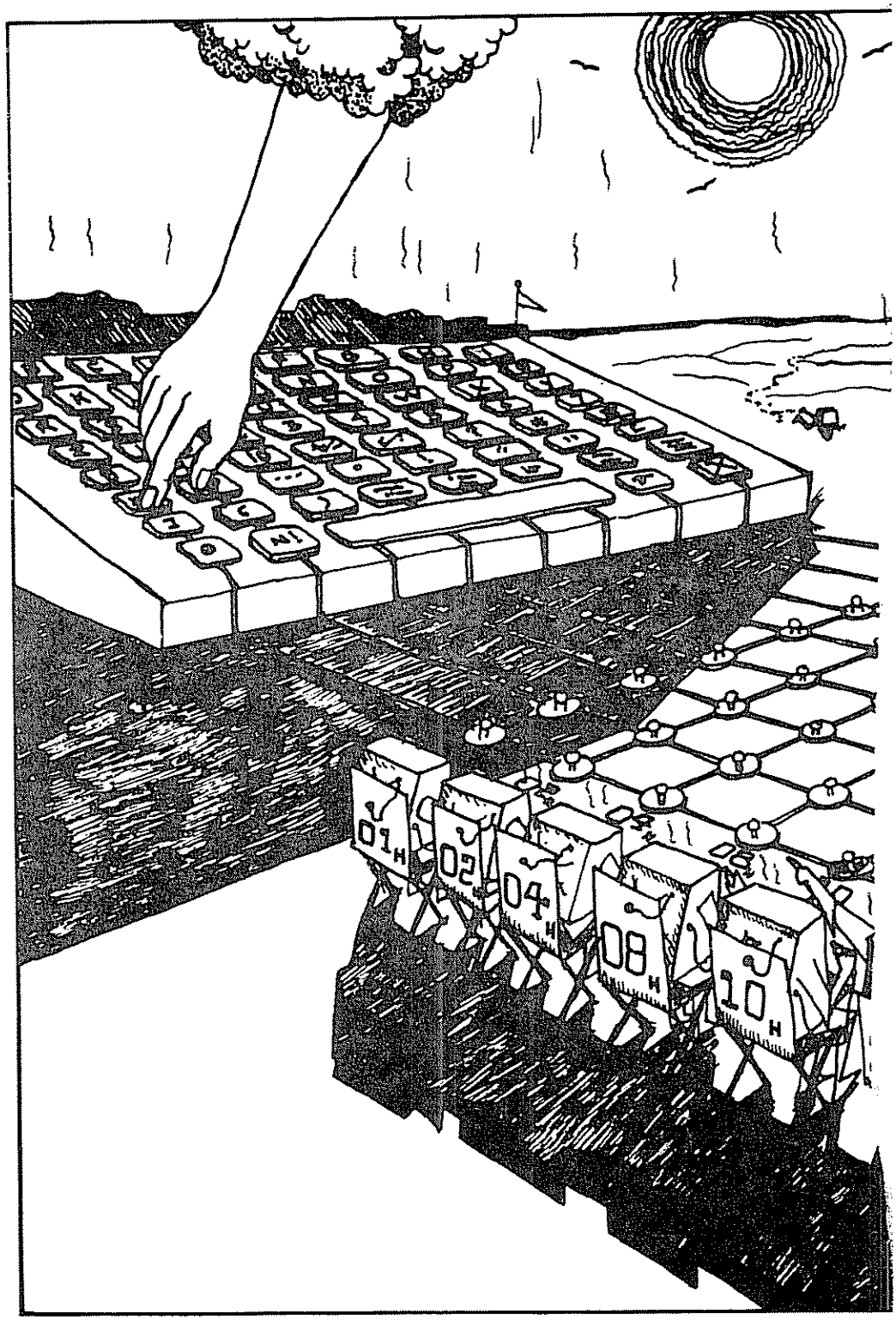
The keyboard is what may be called a "passive" device (it enjoys being stroked...). The mere pressing of a key causes no action in the TRS-80 CPU. The keyboard must be scanned. This scanning is not continuous and is performed only when requested by Level II or a user routine.

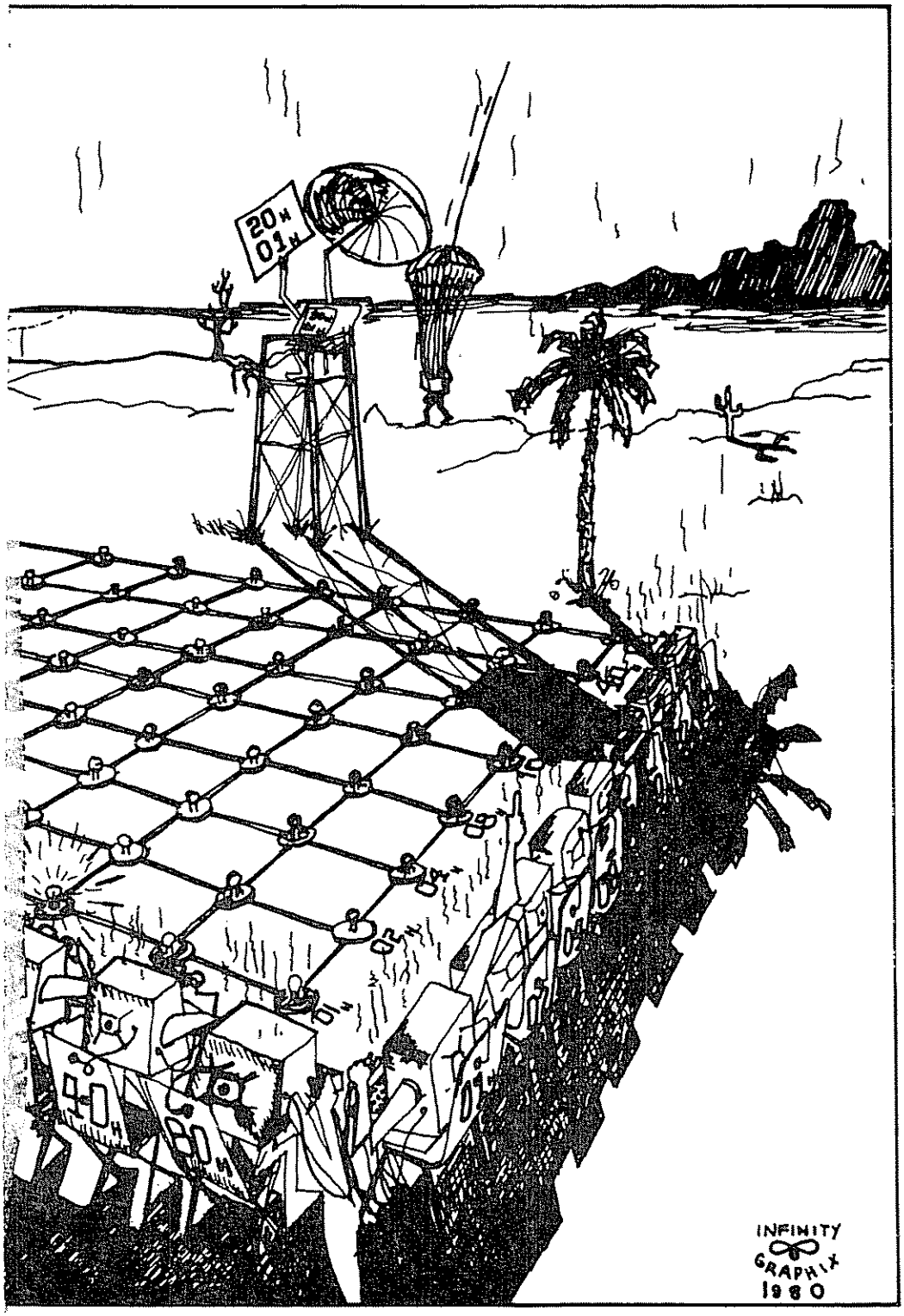
The non-continuous scan causes a problem with keyboard decoding. The problem is this: the system notes that a key is pressed during a scan and returns the matrix value of the character to the caller. The keyboard is re-scanned while the key is still pressed. This scan will also note the depression and return the value of the character. There must be a method of decoding which can distiguish between the new keys pressed and the keys that have already been noted. The Level II ROM uses a table at 4036H-403CH (KBIMAG) to perform this differentiation. KBIMAG is used to store a mirror image of the eight row address values that were read during the last scan.

Here is a possible scenario. The letter 'J' is depressed. The keyboard scan picks up the depression in location 3802H and notes the value, 04H. It stores this value in the KBIMAG table at location 4037H. The rest of the table is set to zero since no other keys are depressed. The 'M' key is then depressed while the 'J' is still held down. The next scan sees at location 3802H a 24H. The scanner then looks at KBIMAG location 4037H and finds a 04H, the value from the last scan. It notes that the 04H key (the 'J') had been depressed on the last pass, or scan. It eliminates this key. The value then becomes 20H. Aha! This is obviously an 'M.' This character is returned to the system as a newly depressed key. The value 24H is then stored into KBIMAG.

The above is an example of how the system determines what key has been depressed on the keyboard. For the assembly code involved to calculate the ASCII value to return, refer to the disassembly [KEYIN at 03E3H].

Hark! I hear a small voice crying out, "What about debounce?" Well, first we must explain what "bounce" is. **Bounce** is that unfriendly phenomenon that occurs when one presses on a key once and gets two characters. Briefly, here is what happens.


01H
02H
04H
08H
10H


20H
01H
INFINITY
GRAPHIX
1980

When a key is pressed, contacts on the inside of the key close and touch. This connection causes the short which ultimately results in the bit being set. Unfortunately, this connection may not be solid. Therefore, the key is pressed and the connection made. The scanner picks this up. Then, the connection is lost while the contacts settle (contacts bounce against each other). The scanner clears KBIMAG since the key is "no longer pressed." The connection settles. The scanner finds this new key. Gee, auto repeat! Not funny, especially to those who are plagued by it.

What is truly amazing about all this is that Radio Shack has actually responded to all the complaints! The new keyboards coming from the Shack have gold contacts. They are easier to use when touch typing since the are slightly curved, have a matte finish, smaller keycap lettering, and respond to a faster stroke. These keyboards are virtually "bounceless."

But what about the poor soul with the old keyboard? There is currently a tape that eases the discomfort of the bounce. In addition, many "debounce" routines have been published in magazines. Choose the most convenient one that works and stick with it. [Debounce routines cause the scanner to pause for a set period of time while the contacts settle before continuing.]

By the way, most people have heard that cleaning the contacts under the keys will decrease bounce. This is true on the OLD keyboards; however, do NOT try to remove the keycaps on the new keyboards. You would probably rather spend the $250 on software.... Also, most of the new disk operating systems have debounce routines build-in.

We need a bit more on scanning before we get to the meat of keyboard interfacing. We have discussed how the TRS-80 uses the eight addresses which have only one bit set in the LSB. Anyone have a guess how we might use the other addresses?

First of all, the TRS-80 ROM does not utilize the other addresses under Level II. But, they can be used from an assembly language procedure. First of all, let's look at the LSB of an address:

C1H = 1100 0001

This LSB has bits 7, 6, and 0 set. This means that a scan at a KEYMEM location with this LSB (i.e., 38C1H) would scan for characters in rows 7, 6, and 0. For example, if the value at 38C1H is 1001 0001 (91H), the keys pressed could be G, D.A, and SFT. Of course, other combinations are possible.

To test whether or not ANY key is pressed on the keyboard, you can scan location 38FFH. Since all keys map to this location (FFH = 1111 1111), if (38FFH)=0 then no key is pressed. You may want to examine the lowercase driver in Appendix B for a method of scanning and decoding which allows auto repeat and specially defined keys.

KEYBOARD DCB

The keyboard DCB format is as follows for standard TRS-80 Level II BASIC operation:

```
                KEYDCB  EQU   4015H          ;KB DCB Location
                        ORG   KEYDCB
4015 01         KBTYP   DEFB  01H            ;Input device
4016 E303       KBDADR  DEFW  03E3H          ;Address KEYIN driver
4018 00         KBCONS  DEFB  00H            ;Keyboard constants
4019 00                 DEFB  00H
401A 00                 DEFB  00H
401B 4B49               DEFM  'KI'
```

To use an alternate keyboard driver, you would have to change the driver address at location 4016H in a manner similar to the following:

```
KEYDCB  EQU   4015H          ;Keyboard DCB
KDBADR  EQU   KEYDCB+1       ;KB Driver address
        LD    HL,KEYDRV      ;Ld address new DRVR
        LD    (KBDADR),HL    ;Store new address
        JP    06CCH          ;Re-enter BASIC
KEYDRV  XOR   A              ;New keyboard driver
                             ; (Locks keyboard)
        RET
```

INTERFACING ROUTINES

The following assembly language interfacing routines should be used to input characters from the keyboard. Use care and note all special considerations listed. Prudence dictates that you also refer to the disassembly at some point. This is necessary to best understand what is happening inside the machine during ROM subroutine execution.

Single Character Scans

The following routine scans the keyboard. If a new key is pressed, the ASCII value of the key is returned. If no new key is found, the routine returns an ASCII null (NUL = 00H).

```
KBSCAN  PUSH  DE          ;Save 'DE'
        CALL  002BH       ;Scan keyboard
        POP   DE          ;Restore 'DE'
        OR    A           ;Set flags
        JP    Z,NOKEY     ;If value returned is
                          ;00H, no key found
        JP    KFOUND      ;A new key was hit.
```

You may also use another routine which does not require you to save 'DE'. However, you must be careful when using this routine under many DOS systems. At location 0358H, this procedure does a CALL to 41C4H. This is what is refered to as a "hook." Under normal Level II, 41C4H contains a C9H, the opcode for RET. Under disk systems, this may be changed to a jump to another routine, possibly causing unpredictable results. Control of this area is left to the programmer.

```
        ;Hook at 41C4H should be considered.
KBDSCN  CALL  0358H       ;Call the scan
        OR    A           ;Set flags
        JP    Z,NOKEY     ;No key found
        JP    KFOUND      ;New key pressed!
```

Single Character Inputs

The following routines scan the keyboard until a key has been pressed and return the value in register 'A.' A possible problem with this looping is the system is "hung" until a key is pressed [Note: The code returned for BREAK is 01H. BREAK is used on most systems to interrupt execution. However, on the TRS-80, the user's program must detect the code for BREAK and operate accordingly.]

```
GETCHR  PUSH  DE            ;Save 'DE'
        CALL  0049H         ;Get a character
        POP   DE            ;'DE' restored
                            ;Char. in 'A'
```

Another routine to input a character saves 'DE' but also has a disk hook at 41C4H (the same hook as KBDSCN, since this routine calls KBDSCN). Be sure to consider this hook when using the following call.

```
        ;Hook at 41C4H should be considered.
GTDCHR  CALL  0384H         ;Get a character
```

* * *

A note about the above routines: each of the routines which scan the keyboard for a character do NOT display the character on the screen; they only return the ASCII value resulting from the scan. If these characters are to be displayed as they are detected, the programmer must specifically CALL another routine. You may want to refer to the section in the next chapter on displaying single characters on the video.

Buffered Line Input

The routine **BUFFIN** can be used to input a programmer-defined maximum number of characters into a buffer. The routine displays each character on the video as it is entered into the buffer. This routine is especially useful since all of the standard TRS-80 control characters can be used. For example, the left arrow performs a backspace and erases the last character. The CLEAR key erases the screen and the buffer. Shift left arrow erases the current line. This is the same routine that is used to input a BASIC text line. The interface is as follows:

```
BUFFIN  LD    B,MAXCHR      ;Input 'B' characters
                            ; into a buffer of
                            ; length 'B'+1
        LD    HL,BUFFER     ;Point to buffer
        PUSH  DE            ;'DE' saved
        PUSH  BC            ;'BC' saved
        CALL  05D9H         ;Get buffer of chars.
        JP    C,BREAK       ;Carry set if BREAK
                            ; hit to end input
        LD    A,B           ;Save number of chars
                            ; in buffer
        POP   BC            ;Restore 'BC'
        POP   DE            ;Restore 'DE'
```

As you will note from the above example, on entry to the BUFFIN routine located in ROM at 05D9H, 'B' contains the maximum number of characters to input into the buffer. Special Note: The buffer MUST BE OF LENGTH 'B'+1 since the ending character [either ENTER (0DH) or BREAK (01H)] is placed into the buffer also. Upon return, the carry flag is set if the input ended with a BREAK and 'B' contains the number of characters in the buffer. 'HL' should point to the beginning of the buffer both before and after execution.

There are other ways to interface with the ROM routines to get input from the keyboard. However, the routines outlined above serve most admirably and require the least amount of preparation before making the CALL. You may refer to the ROM disassembly for other less elegant interfacing procedures that may be used to input from this device.

This concludes the discussion of the keyboard. Now that we have a method available to us to get data from you to the computer, let's look at getting data from the computer to you.

Output: The Video

The video display is the primary output device used on the TRS-80. The screen is "driven" by a **Cathode Ray Tube** (CRT) in the video display. The electronic beam of the CRT travels from the top of the screen to the bottom and from left to right. Each screen consists of 264 scan lines. Of these only 192 scan lines are used for the picture; seventy-two lines are used as upper and lower boundaries. Nothing is ever written or visible within these 72 lines. Each screen consists of either 1024 (400H) or 512 (200H) character locations, depending on whether the machine is in 64-character or 32-character mode. There are 16 character lines, each consisting of 12 scan lines. Each alphanumeric character uses the upper seven of the twelve lines; the five lower lines are used as blank scan lines to provide for spacing. That's enough of the technical trivia--on to the useful stuff.

CHARACTER OUTPUT

We'll cover character output first since it is fairly straight forward; graphics output is more complicated.

When used for character output, the video should be viewed as a simple 16 row by 64 column matrix. Each of these locations corresponds to one of the members of the video RAM area which begins at 3C00H and ends at 3FFFH. This area is referred to as **CRTMEM** (The video is a memory-mapped device addressed at 3C00-3FFFH). Each row in the matrix is 40H entries in length. Therefore, row 1 is at 3C00H-3C3FH, row 2 is at 3C40H-3C7FH, etc. A complete breakdown of the rows by their beginning memory address (begin), middle address (begin + 20H), and last address (begin + 3FH) is as follows:

	Begin	Middle	Last
ROW 1	3C00H	3C20H	3C3FH
ROW 2	3C40H	3C60H	3C7FH
ROW 3	3C80H	3CA0H	3CBFH
ROW 4	3CC0H	3CE0H	3CFFH
ROW 5	3D00H	3D20H	3D3FH
ROW 6	3D40H	3D60H	3D7FH
ROW 7	3D80H	3DA0H	3DBFH
ROW 8	3DC0H	3DE0H	3DFFH
ROW 9	3E00H	3E20H	3E3FH
ROW 10	3E40H	3E60H	3E7FH
ROW 11	3E80H	3EA0H	3EBFH
ROW 12	3EC0H	3EE0H	3EFFH
ROW 13	3F00H	3F20H	3F3FH
ROW 14	3F40H	3F60H	3F7FH
ROW 15	3F80H	3FA0H	3FBFH
ROW 16	3FC0H	3FE0H	3FFFH

The video RAM is written to and read from in the same manner as regular memory. The actual memory consists of 8 static RAMs which do not require refreshing. To display any printable ASCII character on the screen, simply place the code for the character into the video memory. The video generator will do the rest.

Remember, all ASCII characters are not printable. If you were to place in video memory the ASCII value for a non-printing character, a backspace (08H) for example, it would not cause the intended action on the screen. Instead, it would display as an "H" (48H). It is stored in the video memory as a 08H, but the video generator displays on the screen the ASCII equivalent of 48H.

This is the case on most new machines with lowercase capability built in. However, on the early machines (ones without the 8th static RAM for bit 6), the computer uses another method of determining the value of the sixth bit. On these machines, if both bit 5 **and** bit 7 are off, then bit 6 will be high. This means that for values placed into the video RAM of a magnitude less than 20H, bit 6 of the byte will be set (the equivalent of adding 40H). For this reason, you should not use video RAM for data storage.

Example:

01H = 0000 0001 placed at 3C00H on a machine without the bit 6 static RAM becomes
41H = 0100 0001

Now that we have covered the basics of printable character output, let's discuss the TRS-80's graphics.

GRAPHICS OUTPUT

The video screen when it is used for graphics display may be broken down to 6144 positions arranged in a 48 row by 128 column grid. Each position in the grid defines a **pixel** (picture block, sometimes called a graphics block) three dots wide by four lines high. When compared to the character format, the area on the screen which is occupied by one character may be broken down into six pixels (2x3 block). Each of these pixels may be turned on or off independently of the the other pixels. We will number the pixels located in one RAM location in the following manner:

0 1
2 3
4 5

But how do we control each pixel? Let's look at the binary value retrieved from the video memory. This value is from a block with all of the pixels on:

BFH = 1011 1111

Each of the lower 6 bits (b0-b5) correspond to one of the pixels. Bit 6 is unused. Bit 7 is used as the "graphics indicator"; when this bit is on, the lower six bits are used to determine whether a pixel is on or off. As you might have guessed, bit 0 corresponds to pixel 0, bit 1 matches with pixel 1, and so on. Gee, that's pretty logical.

For a complete list of graphics characters and their hex and decimal values, you can refer to Appendix E in the back of the book.

Since bit 7 determines whether the bits are used to define a graphics block or a character block, you cannot have a printable character and a graphics character corresponding to the same address in the video RAM. To display a character, bit 7 must be off. To display graphics pixels, bit 7 must be on. Obviously, we cannot have both.

You may be familar with the SET and RESET instructions in BASIC. Using these commands, you can set ("turn-on") or reset ("turn-off") any individual graphics block on the video screen. The assembly language code for these commands is not as simple as just "set this block" or "reset this block." The software must calculate which of the 1024 locations must be used for an individual pixel. How might this be done?

First of all, the row number (0-47) is divided by three and the integer result is multiplied by 40H (the length a row). This value is added to the column number divided by two. This produces the relative offset of the byte to be used (add 3C00H to produce the address). The remainder of the row division (0, 1, or 2) is used to determine the row within the block and the remainder from the column division (0 or 1) is used to determine the column position. If you are interested in the assembly language code which performs these functions, it is located in ROM from 0132H to 019BH.

Here is a short little routine which demonstrates some of the above concepts:

```
; This assembly language routine will
; scan the video memory, picking out the graphics
; characters, setting the blocks which are off
; and resetting the blocks which are on.
; By repeating the routine again, the original values
; are restored.

FLASH   LD    HL,3C00H      ;Pt. to beginning of
                            ; the screen
        LD    BC,400H       ;1024 locations
CLOOP   LD    A,(HL)        ;Check character from
                            ; the video RAM
        BIT   7,A           ;Test graphics bit
        JR    Z,NEXT        ;If off, not graphics
        CPL                 ;Turn 1's to 0's and
                            ; 0's to 1's
        OR    80H           ;Set bit 7 (graphics)
        LD    (HL),A        ;Place back in RAM
NEXT    INC   HL            ;Next RAM location
        DEC   BC            ;Dec location count
        LD    A,B           ;Get MSB of count
        OR    C             ;Test 'BC' for Zero
        JR    NZ,CLOOP      ;Loop if not done
        CALL  0060H         ;Delay CALL
        JR    FLASH         ;Loop again, flashing
                            ; the graphics on the
                            ; screen.
```

As you will recall, each of the 1024 elements of the video screen can be <u>individually</u> defined as either a graphics character or a printing character; it is determined by the status of bit 7--on for graphics, off for characters. To control each of the pixels in a 6-pixel-block, you can set or reset bits 0-5.

Now that we have the basics of programming the video display, on to the interfacing!


HIGH ORDER
01
02
03
04
05
06
07
08
09
10

VIDEO DCB

The video DCB format is as follows for standard TRS-80 Level II BASIC operation:

```
                CRTDCB  EQU   401DH          ;Video DCB Location
                        ORG   CRTDCB
401D 07         CRTTYP  DEFB  07H            ;Output, Input,
                                             ; and Control Type
401E 5804       CRTADR  DEFW  0458H          ;Address VIDEO driver
4020 003C       CURPOS  DEFW  3C00H          ;Cursor position
4022 00         CURCHR  DEFB  0              ;Character at cursor
4023 444F       CRTCON  DEFM  'DO'           ;Video device name
```

To change the address of the video driver, you could modify the contents of CRTADR (401EH), placing the address of the new driver at that location.

The video driver maintains the position of the cursor on the screen (3C00-3FFFH). The position is stored at locations 4020-4021H in standard L,H format. [L,H format is used for storing WORDS (2 bytes long) in memory in LSB-first, MSB-second order. L,H refers to the order that the register 'HL' would be stored in the instruction LD (ADDR),HL. 'L' would be stored at ADDR; H would be stored at ADDR+1. You can also remember L,H as LOW,HIGH.]

CURCHR is used to save the character that is "under the cursor." As the cursor moves across the screen, it "covers" a character as it moves into the next position. The driver takes the byte stored in the address where the cursor is displayed, stores it in CURCHR, and replaces it with a 5FH (the cursor character). When the cursor continues, the cursor character is replaced by the original contents.

INTERFACING ROUTINES

The following assembly language interfacing routines should be used to output characters and graphics to the video. You may want to refer to the disassembly at some point to firm up your knowledge of the video and its drivers.

Single Character Output

The following routine takes the character in 'A' and outputs it to the video at the cursor position (defined by CURPOS at 4020-4021H). Since we are accessing a video driver by using this routine, we can now specify the special control codes (01-1FH). These codes are NOT placed in the video memory, but cause some cursor or screen action. For example, if we send the backspace character to the video driver, the cursor will backspace one character position and erase the previous character (unless it is already at the beginning of the screen). For a full list of these codes, refer to Appendix D.

```
CRTBYT  PUSH  DE          ;Save 'DE'
        CALL  0033H       ;Output the character
                          ; stored in 'A'
        POP   DE          ;Restore 'DE'
```

The following interface is probably the best to use since it maintains the cursor position indicator at 40A6H (CRTPOS). This is the routine used most often by the ROM. The 'DE' registers are saved by this routine also. The interface is as follows:

```
CRTOUT  LD    A,CHAR      ;Load display char.
        CALL  033AH       ;Output the char.
                          ;CRTPOS(40A6H) is up-
                          ; dated automatically
```

A routine that is often used in conjunction with the above interface is POSIND, which returns the current cursor position on the line in the 'A' register. This routine compensates for double width characters.

```
POSIND  CALL  0348H          ;Determine pos on CRT
        LD    (40A6H),A      ;Store position in
                             ;  CRTPOS
```

Remember, the above routines can be used to send control codes, printing characters, graphics characters, and the space compression codes. The value sent to the driver is interpreted by it, and the request is handled accordingly.

Clearing The Screen

The easiest way to clear the screen is to make one simple call to the CLS routine as shown below:

```
CLS     PUSH  AF             ;Save 'AF' if need be
        CALL  01C9H          ;Clear the screen
        POP   AF             ;Restore 'AF'
```

Note: When the screen appears clear, it is actually full of spaces (20H's).

SET, RESET, and POINT

It is not easy to interface to the ROM routines SET, RESET, and POINT due to several interactions with other ROM routines. However, with the information given above regarding the workings of the video and with the ROM routines at 0132-019CH as a guide, it should be easy to write a routine which better suits your needs. If you wish to use the routines, you may refer to our modification of the SET, RESET, and POINT sections in Appendix F.

Input From Video

As mentioned in chapter 1, it is possible to input from the video. By performing a CALL to INBYT when the video DCB has been specified, the character at the cursor position on the screen is returned to the user as follows:

```
        LD    DE,401DH      ;Specify video DCB
INBYT   CALL  0013H         ;Get byte from cursor
                            ; into 'A' register.
```

The value retrieved depends on whether the cursor is on or off. If the cursor is on, the value which is stored in CURCHR is returned in register 'A'. If the cursor is off, 'A' will contain the value in the video RAM address specified by the cursor location on screen (CURPOS).

The video display is used for most output; however, there are times when we would like to print output from our TRS-80. In the next chapter we will discuss interfacing a parallel printer to a TRS-80 Level II.

Output: The Printer

The standard Level II TRS-80 comes with a keyboard unit which contains the CPU and the major processing hardware. In addition to the keyboard, one is supplied with a video display and a tape unit. This set-up works well, to a point. A major drawback is that to take data from one place to another in a form that can be read by human eyes, you have to be able to get a printout of the information on paper. For this purpose, most owners have purchased or are considering the purchase of a printer. In this chapter, we will discuss assembly language control of a parallel printer.

When the TRS-80's first appeared, the owner had to purchase an Expansion Interface for $299 dollars before he could attach a printer to the Level II system. Recently, Radio Shack has added new cables to their line which allow the direct connection of a parallel printer to a TRS-80 Level II. [For the exact catalog number of the cable for a specific printer, you can contact Radio Shack or refer to their catalog. Most printers will use either 26-1411 or 26-1416.

The ROM has a driver built in to support a parallel printer addressed (memory-mapped) at **LPTADR** (37E8H). The driver uses ROM at 058D-05D8H. It is a very simple driver, but it does work for most Centronics-type parallel printers. For a more sophisticated printer driver, you may refer to the assembly language listing in Appendix H.

Why was the parallel interface chosen? In the words of Radio Shack, "This interface type was chosen because it is a widely used industry standard, is reliable and is easily implemented." This statement is essentially correct. The parallel interface for the TRS-80 is very reliable. For this reason, one should probably purchase one of the many parallel printers on the market, by-passing the serial printers. Many owners of serial printers have found that using the RS232-C and having to load a driver every time they wish to print anything to be frustrating. The ROM has a built-in parallel driver; a separate driver is needed to operate a serial printer. An application in which a serial printer is necessary is one that requires

that input from an external printer be entered into the computer via the RS232-C.

LINE PRINTER STATUS

The line printer port (37E8H) has four status bits. These bits show the conditions busy, out of paper, device selected, and no fault. The corresponding bits are as follows:

bit	Status if bit set
7	BUSY
6	OUT OF PAPER
5	DEVICE SELECTED
4	NO FAULT

BUSY means that the printer is unable to accept any data. This condition can occur when the printer is off, has a full buffer, is printing a line, is out of paper, or is just not prepared to print because of some physical condition such as no ribbon or the top is up.

OUT OF PAPER is self-explanatory. When the printer switch (if one is present) detects that the paper is out, the printer sets this bit. The expansion interface takes this status and ORs it with the BUSY bit. Therefore, if the printer is out of paper, it will also be busy. However, contrary to RS documentation, if the printer is shown as busy, it will not necessarily show as out of paper.

DEVICE SELECTED is incorrectly documented in Radio Shack literature. This bit is used to show whether the printer is ONLINE (connected and in remote mode, ready to receive data from the computer). It is used, along with NO FAULT by the printer driver in ROM, although the latest documentation out of Tandy says that it is not.

NO FAULT is a bit that signifies whether an error condition is present in the printer (paper out, etc.). It is a logic-1 flag, as is DEVICE SELECTED, which means that it must be on to signify that the printer is ready to receive data.

The routine in ROM which checks for printer ready is as follows:

```
PSTATU  LD    A,(37E8H)     ;Get printer status
        AND   0F0H          ;Clear low bits
        CP    30H           ;If ready, Z flag set
        RET
```

Now, let's look at the format of the printer DCB before we get to the interfacing.

PRINTER DCB

The format of the line printer DCB as it is initially set by the Level II bootstrap is as follows:

```
4025 01    LPTTYP  DEFB  06H           ;Printer=Output
                                        ; and control codes
4026 8D05  LPTADR  DEFW  058DH         ;Driver address
4028 43    LPTLPP  DEFB  67            ;Lines/page
4029 00    LPTLCT  DEFB  0             ;Line counter
402A 00    LPTCON  DEFB  0             ;Printer constants
402B 5052          DEFM  'PR'
```

Radio Shack incorrectly set the number of lines per page at 67 in ROM. The correct setting should be 66 for standard 8.5 x 11" paper. If the ROM driver is to used to skip to the top of page upon receipt of a control-L <FF=0CH>, you should correct this setting using a POKE 16424,66. It would be nice to change this value permanently, but all DCB values are copied from ROM to RAM at power-up.

The line counter is maintained by the driver. It is incremented once after every carriage return or linefeed. This value is used in conjunction with the number of lines per page to determine how many linefeeds should be issued to advance to the top of the next page when a formfeed is received.

INTERFACING ROUTINES

The following routines can be used to access the parallel printer driver in ROM. Remember, these routines suffer from the same problem as BASIC when using the printer; if there is not a printer present, the system will "lock-up." To avoid this, you may want to create a "NULL DRIVER" if the printer is not present. In assembly language, the format is as follows:

```
NOPRNT  LD    A,0D0H        ;LSB of a RET inst.
        LD    (4026H),A     ;Change driver addr.
        RET                 ;Printer off
```

To get it back:

```
PRNTON  LD    A,8DH         ;LSB of driver addr.
        LD    (4026H),A     ;Restore old driver
        RET                 ;Printer now on.
```

In BASIC:

```
1000  POKE 16422,208        'Printer off
2000  POKE 16422,141        'Printer on
```

By turning the printer into a null device, you will not lock-up the computer if the printer is not present. A better method is to write a small lead-in routine to the printer driver which checks the status of the printer, and if the user holds down the <CLEAR> key, do not send the byte to the printer driver. A possible routine follows:

```
        ORG   4026H
        DEFW  CKPRNT        ;Set new driver
        ORG   7FE0H         ;Possible ORG addr.
CKPRNT  LD    A,(37E8H)     ;Get status
        AND   0F0H          ;Mask low order bits
        CP    30H           ;Ready?
        JP    Z,058DH       ;Goto printer driver
                            ; if printer ready
        LD    A,(3840H)     ;Check for clear
        BIT   1,A           ;Test for <CLEAR>
        RET   NZ            ;Return if pressed
        JR    CKPRNT        ;Loop until ready
```

Single Character Output

The following routines output the character stored in register 'A' to the printer. Remember to consider system lock-up when using these routines.

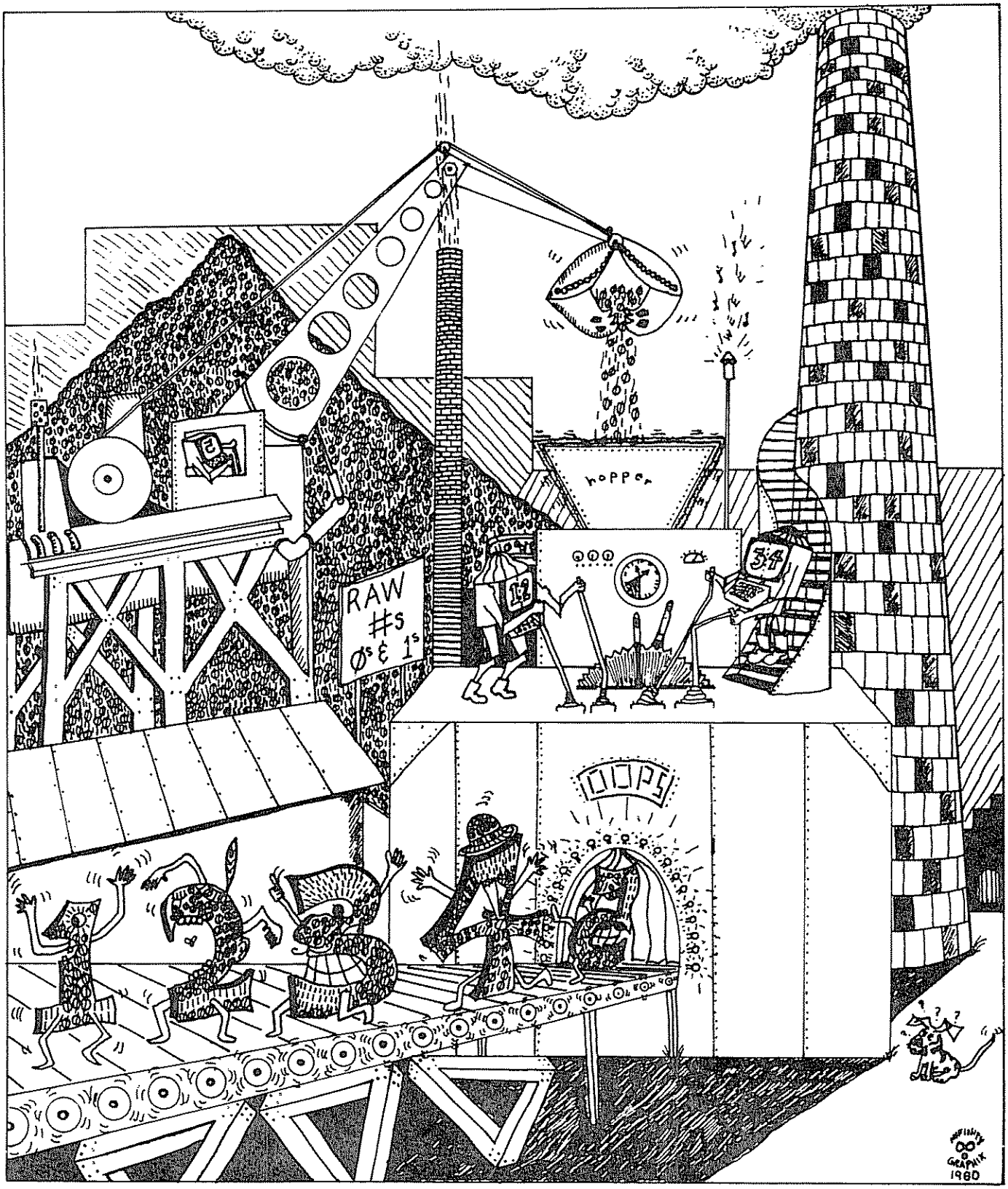
```
        PUSH  DE          ;Save 'DE'
LPTBYT  CALL  003BH       ;Output 'A' > printer
        POP   DE          ;Restore 'DE'
```

This next routine also saves the 'DE' registers and maintains the current line position (number of characters sent to the printer in the current line) at **LPTPOS** (409BH). This is the routine used by BASIC which changes all linefeeds to carriage returns.

```
        LD    A,CHAR      ;Load print character
LPDCHR  CALL  039CH       ;Output to printer
```

These are the only two ROM routines which specifically support the printer. The printer is one device for which you will probably want to use a more sophisticated driver. It would be wise to refer to the printer driver in ROM for hints and programming techniques.

We have now discussed the keyboard, video, and printer. Only one more device remains. In the next chapter, we will fully discuss the mysteries of the tape unit, how to use the ROM tape routines, and how to write your own tape driver.


RAW
#S
Ø$ € 1$
hopper
OOPS
34
GRAPHIX
1980

Input/Output: Tape

To the first-time user, the tape unit is the device which causes the most confusion and frustration. All we know when we buy a TRS-80 is that if we type CSAVE, our program (which we know is stored as numbers) is converted to sound on cassette tape. Then, we type CLOAD and sometimes it loads correctly, and sometimes it doesn't. Furthermore, we buy a program, and after 27 tries at getting the proper volume, we give up hope. Is it the tape? Is it the recorder? Is it the computer? We don't even know what is happening! This chapter is designed to answer these questions and to give a complete explanation of the tape unit. After reading the following material, you still may not be able to load that tape, but at least you will know some of the causes.

TAPE HARDWARE

As a hardware device, the tape uses a latch (see explanation below) which is accessed through Z80 port 0FFH (255). The use of the various bits depends on whether the operation is an OUT or an IN. Let's cover OUTput first.

Upon output to port 0FFH, the hardware takes the upper four bits and discards them. The lower four bits are used to control the video format, the cassette motor, and the output voltages as follows:

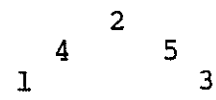
Bit	Output to port FFH use
3	Video Display Mode Select 1=32 char./line 0=64 char./line
2	Cassette motor relay 1=ON 0=OFF
1	CASSOUT B Signal
0	CASSOUT A Signal

The output signals defined by CASSOUTs A and B are defined as follows:

CASSOUT A	CASSOUT B	Output signal	
0	0	No signal	(0.46V)
1	1	Low signal	(0.00V)
0	1	Low signal	(0.00V)
1	0	High signal	(0.85V)

Input from this port uses only the two highest bits. Bit 7 is used for cassette input (the decoded CASSIN signal is latched here until the software clears it). Bit 6 is used to indicate the Display Mode (0 = 32 char., 1 = 64 char.). The reset of the bits are set to 1's.

The pinouts (for what each pin is used) are not necessary for using the cassette tape, but are provided for informational purposes. If you look at the DIN plug, facing the pins toward you, they are numbered from left to right:



Pin	Signal-Name	Description
1	MOTOR ON/OFF	Controls tape motion
2	GROUND	Signal ground
3	MOTOR ON/OFF	Controls tape motion
4	CASSIN	READ signal from tape
5	CASSOUT	WRITE signal from TRS-80

One other bit of hardware is used with the cassette, but only if you have an expansion interface. If you have one, you may define select between the two possible cassette drives by outputting to 37E4H. If you wish to select drive 1, use:

```
XOR   A              ;Reset bit 0
LD    (37E4H),A      ;Select drive 1
```

To select drive 2, use:

```
LD    A,01H          ;Set bit 0
LD    (37E4H),A      ;Select drive 2
```

Now that we know the hardware, let's look at the software that is used to control it.

TAPE SOFTWARE

As stated in chapter 1, the tape unit does not use a DCB. It does have one RAM byte which is used as a status flag. It is CSTATU, and is located at 403DH. It is a mirror image of the last byte OUTput to port FFH. It is used as a flag to determine whether the cassette is on and whether the machine is in 32-character or 64-character mode. It is also used to clear the **Cassette Flip-Flop** (CFF). Let's look at how information is written to tape.

Write Data

Data is written to the tape one bit at a time. For each bit of data, there is one clock bit which is used to synchronize the software. One bit is written to tape by sending a high signal (A=1,B=0) followed by a low signal (A=0,B=1). [The bits are written as a single cycle of a square wave with a period of 265 microseconds (a frequency of just under 4 kHz)]. This is followed by the "no signal" for about 735 microseconds. The total time is about one millisecond. This is for one bit. This bit does not necessarily represent data, just a bit. Well then, how do we get a bit of **data** on the tape?

First, we send the clock bit. This bit is going to be used by the read routine to signal that a data bit follows.

The next operation depends on whether the data bit to be written is a one or a zero. If it is a zero, we simply delay for one millisecond. If it is a one, we write another bit to the cassette. Simply, the presence of a pulse on the tape following a clock pulse denotes a 1; if no pulse is present, the data bit is a 0. Data is written to the tape as a constant stream of clock bits, followed by either a pulse to mean a one, or no pulse to mean a zero.

As one should see, to write a byte, the software must repeat above procedure eight times, once for each bit. For a byte, the bits are written in decreasing order (7,6,5,...,0). Also, since it take 2 milliseconds to write or read each bit of data, the tape can process a total of 500 bits per second or 62.5 bytes/second.

All bytes for all the different tape formats supported by the ROM are written in this manner. The problem is reading what has been written. Unfortunately, this is where the errors come in....

Read Data

The reliability problems associated with the cassette come from many areas. First of all, the cassette recorder is of very low quality. And, as with all recorded material, the media plays a very important role; cheaply made tapes do not work as well as better quality ones. [Note: the actual price you pay has little to do with the quality. For example, some stores discount high quality tapes which gives them a low price tag, many times less than "garbage" tapes.] But the biggest problem is timing. In the older ROMs ("MEMORY SIZE?"), the section to read a bit from the cassette did not wait long enough for the pulse to be detected before continuing. Therefore, data bits that should have been 1's were read as 0's. The new machines ("MEM SIZE?" have increased the timing delay by about 100 microseconds for the data bit (if present) to be latched.

We've mentioned "latch" here, but you haven't been told what one is. Electronic signals such as the pulse read from the cassette do not "stay around." They are present for only a brief time (brief even in relation to a microsecond). Therefore, it must be latched. If the signal goes high, it sets the latch which stays high until it is reset by the software. The reset of the cassette latch is done by doing an OUT to the cassette port. The

ROM routines use CSTATU and simply output its value. The routine CLRCFF at 021EH performs this function after every bit is read.

In addition to the above problems, tape reading errors can be caused by a bit being dropped (lost) from the tape. If the tape is exposed to a magnetic field, it will almost certainly lose at least one bit. Unfortunately, there is no way to recover such a loss.

The interfacing to the tape read and write routines are given after the discussion of tape formats. A very good reference for the actual decoding of the audio signal is given in the TRS-80 Technical Reference Manual (Catalog 26-2103).

TAPE FORMATS

There are four different tape formats that are used by the computer. These are the BASIC language source tapes, the SYSTEM tapes, BASIC langauge data tapes, and Assembler source tapes. There are of course other tape formats written by non-Radio Shack suppliers of software. These will not be discussed here.

In order to assure synchronization at the beginning of the tape, a leader of 255 bytes of zero is written. Since a zero byte will only have clock bits occuring on the tape, by waiting until a pulse is detected, the tape reading program is guaranteed to find a clock bit rather than a data bit. Following this synchronization, the TRS-80 will wait for a clock bit before each data bit. This compensates for motor speed fluctuations. This same leader is written out at the beginning of each tape format described in this section.

While reading the leader, the tape program shifts each new bit into the accumulator at the low end and checks the new value in the accumulator. This is done until 'A' contains the value A5H (10100101 - a symmetrical bit pattern). This synchronization byte is used to mark the beginning of the data bytes on these tape formats.

BASIC PROGRAMS

The first tape type we will discuss is the one produced by the BASIC command CSAVE. Following the A5H

there are three bytes of D3H to indicate the tape type. This is followed by the single ASCII character that was specified following CSAVE (the name of the program). This in turn is followed by a series of program lines in the following form.

First are two bytes representing a pointer to the memory location that the line following the present one occupied at the time of CSAVE. This is followed by two bytes that contain the binary representation of the line number. Next come a variable number of ASCII bytes that represent the actual text of the BASIC statement line with compressed keywords. The end of the line is marked with a byte of zero (00H). There is one of these constructs for each line of the BASIC program. To indicate the end of the program, there are two bytes of zero.

Note that no error checking is performed! Therefore, one has no way of knowing whether the tape was loaded correctly except by doing a CLOAD? which takes up a lot of time if it is a long program. This format is definitely a drawback. Microsoft should have used a diferent format.

If you have a disk system, you should note that the addresses placed on the tape are those that were in the program at the time it was CSAVE'd. Since the BASIC interpreter places the line in the appropriate memory location, you can read tapes under Disk BASIC that were written under Level II BASIC and vice versa. However, CLOAD? will always report a faulty load if transfering from one BASIC to the other.

DATA TAPES

The data tapes produced by BASIC are quite simple. After the leader and sync byte, the data is present as ASCII with a leading blank or minus sign and a trailing blank for numeric data. Individual items are separated by commas. Each PRINT# statement generates a new leader and sync byte.

Again, no checksum is used! If you are doing BASIC programming and use the tape to store and retrieve data, be sure to add some error checking or data validity routines. If you don't, you may be very surprised by the results.

SYSTEM TAPES

The SYSTEM tapes have a much different format than either of the preceeding. Following the leader and sync byte, there is a 55H byte that is the header for the tape name. The name field is the next six bytes in which the ASCII name is left justified and padded with blanks if it is less than six characters long. Following the name are a variable number of records which can be of variable lengths. The records are formed as follows.

The first byte of each record is a 3CH. This is followed by a byte that contains the length of the record (a zero means the record is 256 bytes long). Next come two bytes in LSB/MSB order which is the starting memory address where the current record should be stored. These are followed by the data bytes as specified by the length. The final byte is the checksum which is the sum of all the data bytes and the memory address (if this byte does not match the calculated value, the SYSTEM loader in ROM will place a 'C' in the upper corner, but it continues to load records. This at least gives some visual indication that the tape is bad). This format is repeated as many times as is necessary to load all the data. The end of the tape is signaled by a three byte trailer starting with 78H. This is followed by two bytes in LSB/MSB order which contain the transfer address of the program. You may wish to refer to the alternate SYSTEM loader in Appendix C.

ASSEMBLER SOURCE TAPES

These tapes start with a D3H byte following the leader and sync byte. This is followed by a six byte ASCII name field. The text lines follow.

Each line has a five byte line number in ASCII format with bit 7 set to differentiate these line numbers from other numbers in the text. Next there is an ASCII space (20H). This is followed by the ASCII text and finally by a carriage return (0DH). This format is repeated for each line. The end of the tape is indicated by a 1AH. Once again, no error checking is performed.

INTERFACING ROUTINES

The following 15 ROM interfacing examples cover all the reading and writing of data to the tape unit, and includes the control of the cassette recorder.

Cassette Recorder and Latch Control

The following routine turns the cassette drive 1 on. First it does a check on the character at (HL) to see if it is a pound sign (#). If it is not, drive 1 (which is selected by outputting a 00H) is turned on. If it is a pound sign, the number starting at (HL+1) is converted to an integer. A syntax check is then performed for a comma. Then the integer value is converted to a drive number. If the driver number is invalid, Illegal Function Call results. Take care when using this routine that HL does not point to a stray pound sign, or you may find yourself in BASIC.

```
        PUSH   HL            ;Save 'HL'
        LD     HL,0          ;Point HL to safe loc
CTON    CALL   01FEH         ;Turn cassette drive
                             ; on.
        POP    HL
        RET
```

This routine defines the drive in register 'A' (Drive number -1) by outputting to the cassette select latch. It then turns on the drive.

```
        LD     A,1           ;Drive 2
DEFDRV  CALL   0212          ;Define drive 2.
                             ;Drive on
        RET

        LD     A,0           ;Drive 1
DEFDRV  CALL   0212          ;Define drive 1
        RET
```

This routine turns the cassette off. The accumulator is used and must be saved.

```
        PUSH   AF            ;Save 'AF'
CTOFF   CALL   01F8H         ;Cassette off
```

```
        POP   AF          ;Restore 'AF'
        RET
```

The following code clears the cassette flip-flop by simply outputting the value stored in (CSTATU).

```
        PUSH  HL          ;Save 'HL'
        PUSH  AF          ;Save 'AF'
CLRCFF  CALL  021EH       ;Clear CFF
        POP   AF          ;Restore 'AF'
        POP   HL          ;Restore 'HL'
```

The CLRCFF routine uses the STATFF routine at 0221H to make the change. STATFF takes the value in (CSTATU), ANDs it with 'H' (to reset any bits) and ORs it with 'L' (to set any bits). It then saves the result in (CSTATU) and outputs the result to port FFH. This routine is used to change the voltage levels, turn the drives on and off, and turn on or off display modes. The values to load into HL are as follows:

```
HISIG   LD    HL,0FC01H   ;High signal
LOSIG   LD    HL,0FC02H   ;Low signal
NOSIG   LD    HL,0FC00H   ;No signal

OFFC    LD    HL,0FB00H   ;Cassette off
ONC     LD    HL,0FF04H   ;Cassette on

RLATCH  LD    HL,0FF00H   ;Clear CFF

M64     LD    HL,0F700H   ;64-char mode
M32     LD    HL,0FF08H   ;32-char mode
```

To use the routine, select the appropriate value of HL (STATHL) from the above list and make the CALL:

```
        PUSH  HL          ;Save 'HL'
        PUSH  AF          ;Save 'AF'
        LD    HL,STATHL   ;HL Command Selection
STATFF  CALL  0221H       ;Change status
        POP   AF          ;Restore 'AF'
        POP   HL          ;Restore 'HL'
```

Tape Reads

The following routines are used to read from the cassette. Included are routines used by the read-tape ROM sections.

This interfacing routine will turn the cassette drive on using CTON and read the leader until a sync byte (A5H) is found. Then, the stars are placed in the upper righthand corner of the screen at locations 3C3EH-3C3FH.

```
        PUSH  HL          ;Save 'HL'
        LD    HL,0        ;Save (HL) for CTON
CTONRL  CALL  0293H       ;Drive 1 on,
                          ; read leader
                          ; put stars
        POP   HL          ;Restore 'HL'
```

The CRLDR routine is part of CTONRL but the entry to this routine is after a drive has been defined. Use the following linkage:

```
        PUSH  DE          ;Save 'DE'
        EX    DE,HL       ;HL to DE
        LD    HL,RETADR   ;Load a return addr.
        PUSH  HL          ;Save return address
        PUSH  DE          ;Save old HL.
                          ; will be restored by
                          ; routine
CRLDR   JP    0293H       ;Read leader for sync
                          ; put stars
RETADR  POP   DE          ;Restore 'DE'
```

The ROM does have a routine to put the stars in the corner, but if you do not use one of the above routines to read the leader, you must put them there yourself using:

```
CSTARS  PUSH  AF          ;Save 'A'
        LD    A,'*'       ;Star
        LD    (3C3EH),A   ;Put first star
        LD    (3C3FH),A   ;Put second star
        POP   AF
```

This routine reads the data bit following a clock bit and shifts it into the low order end of 'A'.

```
        PUSH  HL          ;Save 'HL'
CRBIT   CALL  0241H       ;Read a bit into 'A'
        POP HL
```

The linkage that follows reads a byte from the tape into the 'A' register.

```
CRBYTE  CALL  0235H       ;Read a byte
```

To change the star in the corner from a blank to star or from a star to blank, simply:

```
        PUSH  AF          ;Save 'AF'
CSTAR   CALL  022CH       ;Change star
        POP   AF
```

Tape Writes

The following routines write data to the tape unit.

To turn the cassette on using CTON and write a leader, use the following interface:

```
        PUSH  HL          ;Save 'HL'
        LD    HL,0        ;Safe (HL)
        PUSH  AF          ;Save 'AF'
CTONWL  CALL  0284H       ;Write leader & sync
        POP   AF          ;Restore 'AF'
        POP   HL          ;Restore 'HL'
```

If the drive has already been turned on, you can use:

```
        PUSH  AF          ;Save 'AF'
CWLDR   CALL  0287H       ;Write leader & sync
        POP   AF          ;Restore 'AF'
```

To write a bit, you may use the following code:

```
        PUSH  HL
```

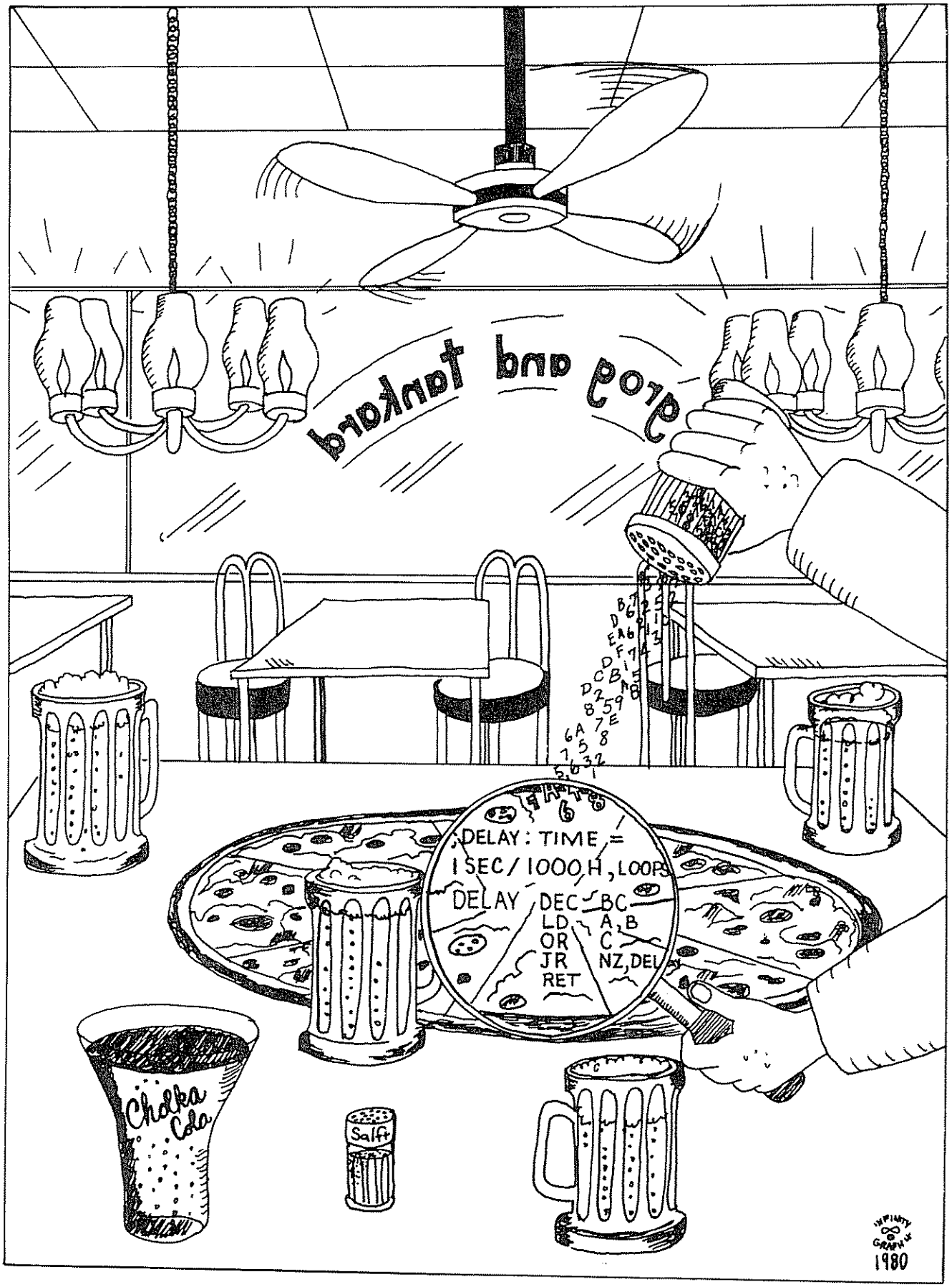
```
        PUSH  BC
        PUSH  DE
        PUSH  AF
CWBIT   CALL  01D9H         ;Write a bit
        POP   AF
        POP   DE
        POP   BC
        POP   HL
```

To write the byte stored in register 'A' to the cassette tape, call the CWBYT routine at 0264H. All registers are saved:

```
CWBYT   CALL  0264H         ;Write a byte to tape
```

A sister routine at 0261H (CW2BYT) simply writes the byte to tape twice using CWBYT.

This completes our discussion of the individual I/O units of the TRS-80. The ROM disassembly of the routines follows in chapter 6. Additional information on I/O routines may be found in chapter 7 and chapter 8.


Chalka Cola
Salft
,DELAY: TIME,=
1SEC/1000,H,LOOPS
DELAY DEC BC
LD A,B
OR C
JR NZ,DEL
RET
1980

ROM Disassembly

In this chapter, you will find a commented disassembly of the Radio Shack Level II ROM input and output routines. However, a few important points must be made about this disassembly.

First of all, the ROM code is the property of Microsoft and is protected by their copyright. For this reason, it is impossible to provide a complete disassembly of their code without violating their rights. For this reason, the publisher has decided to provide the hex addresses of the instructions, the operators, and the extensive comments. The hex object code and the operands are omitted.

If you are an owner of a TRS-80, you are able to procure the full disassembly by using one of many machine language disassemblers available commercially, or if you have purchased Volume I, you can use the BASIC language disassembler found in Appendix C of that volume. Space has been provided so that operands can be written in next to the operators to provide a commented listing that can be used for reference.

Secondly, since a full interfacing guide is provided in the earlier chapters, it is unnecessary to refer to this listing in order to interface with the routines. Nevertheless, this chapter may serve as a useful tool when programming I/O on the TRS-80.

```
                ;**************************************************
                ;**************************************************
                ;** Radio Shack Level II BASIC ROM as Commented by **
                ;** Insiders Software Consultants, Inc. This is a **
                ;** pseudo-disassembly which does not contain op- **
                ;** codes or operands to protect the proprietary **
                ;** source code of Microsoft, the original author **
                ;** of the BASIC Interpreter.                     **
                ;**************************************************
                ;**************************************************

     ;**********
     ;* CBOOT: Upon initial power-up, the execution of an illegal opcode,
     ;*        or a JP to location 0000H, the machine will boot.
     ;*        This entails a reset of certain key RAM locations,
     ;*        a reset of I/O devices such as the printer and tape,
     ;*        and a re-entry into the BASIC interpreter with all
     ;*        pointers reset.
     ;**********
0000            CBOOT   DI                      ;Disable interrupts during boot.
0001                    XOR                     ;Reset Accumulator
0002                    JP                      ;Exit to COLDSTART routine

0005                    JP                      ;Unused in Level II
                                                ;In some machines, CALL 5 is used
                                                ;as a procedure request entry
                                                ;point, similar to CP/M. In the
                                                ;TRS-80, it is not used.

     ;**********
     ;* RST08: This RST is used by the parser to check the syntax of the BASIC
     ;*        program. This RST vectors to 4000H which in turn jumps to
     ;*        location 1C96H under normal Level II operation.
     ;**********
0008                    JP                      ;RST 8 Vectors to 1C96H IN LII

     ;**********
     ;* WHERE: This routine is used to locate the execution location in memory.
     ;*        For example, if one wanted to determine whether the current
     ;*        routine is running in high memory after a relocation, one could
     ;*        CALL WHERE and then check the contents of the HL register.
     ;*        (HL contains the address of the next instruction after
     ;*        CALL WHERE).
     ;**********
000B            WHERE   POP                     ;Locate self in MEM
000C                    JP

000D            DBOOT   JP                      ;JP Disk Bootstrap
```

```
                ;**********
                ;* RST16: This RST is used by the parser to process through the text
                ;*        buffer finding the next character to be processed. Returns with
                ;*        the character in the buffer pointed to by HL. Carry flag
                ;*        set on ASCII 0-9. Zero flag set on colon (:) or binary zero.
                ;*        Skips tabs, linefeeds, and spaces.  This RST vectors to 4003H
                ;*        which in turn vectors to 1D78H.
                ;**********
0010            RST16   JP                      ;RST 16 Vectors to 1D78H

                ;**********
                ;* INBYT: Input a byte from a device. At this point, DE has been loaded
                ;*        with the DCB location. BC is saved. DE is lost. B is loaded with
                ;*        the operation type flag which is compared with the DCB to check
                ;*        for valid I/O requests.
                ;**********
0013            INBYT   PUSH                    ;Input a byte from a device
0014                    LD                      ;B with DCB type (Input)
0016                    JR                      ;JP I/O Driver

                ;**********
                ;* RST24: This RST is used to compare the values contained in the HL and
                ;*        DE registers. (16 bit compare)  If HL=DE, Zero set. If HL<DE
                ;*        then Carry set.  The 'A' reg is lost.
                ;*        RST24 vectors to 4006H and then to 1C90H.
                ;**********
0018            RST24   JP                      ;RST 24 Vectors to 1C90H IN LII

                ;**********
                ;*OUTBYT: This routine is used to output a byte to a device specified by
                ;*        the DCB pointed to by DE upon entry. DE is lost during
                ;*        execution.
                ;**********
001B            OUTBYT  PUSH                    ;Output a byte to a device
001C                    LD                      ;LD B with DCB type (Output)
001E                    JR                      ;JP I/O Driver

                ;**********
                ;* RST32: This RST is used to determine the data type of the current
                ;*        value in FPA1 (See Volume I).
                ;*        Flags set are as follows:  M=Integer, Z=String,
                ;*        PO=Single-precision, NC=Double-precision.
                ;**********
0020                    JP                      ;RST 32 Vectors to 25D9H

0023            CTLBYT  PUSH                    ;Output a control byte to a device.
0024                    LD                      ;(Unused in LII)
0026                    JR                      ;JP I/O Driver
```

```
                ;**********
                ;* RST40: This RST is used under DOS to request processes and overlays to
                ;*        be loaded.  If not executing under DOS, this RST may be used by
                ;*        the assembly language programmer by placing a JP at
                ;*        location 400CH.
                ;**********
0028            JP      RST40V              ;Jump to DOS command processor

                ;**********
                ;*KBSCAN: Scan the keyboard using the routine specified by the keyboard
                ;*        DCB at location 4015H. If a key is pressed, its ASCII value is
                ;*        returned in the 'A' register. If no key, a binary zero is
                ;*        returned.  DE is lost when calling this routine.
                ;**********
002B            KBSCAN  LD                  ;LD DCB Location
002E                    JR                  ;JP to INBYT
                ;**********
                ;* RST48: This RST is used under DOS as the DEBUG breakpoint. It may be
                ;*        re-defined by the assembly language programmer using level II
                ;*        only by placing a JP at location 400FH
                ;**********
0030                    JP                  ;RST 48 is used as DEBUG Breakpoint

                ;**********
                ;*CRTBYT: Displays the ASCII value in register 'A' on the video display at
                ;*        the current cursor position as stored in locations 4020-4021H in
                ;*        the Video DCB block. DE is lost during call.
                ;**********
0033            CRTBYT  LD                  ;LD DCB Location (401DH)
0036                    JR                  ;JP to OUTBYT

                ;**********
                ;* RST56: This RST is used in interrupt mode 1 (IM 1) under DOS. It should
                ;*        not be re-configured.  After an interrupt, execution begins at
                ;*        location 4012H (which is a jump to a handling routine.)
                ;*        The user could place a clock interrupt handling routine at this
                ;*        location or another routine which may run off an interrupt
                ;*        generated by a non-standard peripheral or the clock in the
                ;*        Expansion Interface if one is present.
                ;**********
0038             RST56  JP                  ;RST56 -- Interrupt handler

                ;**********
                ;*LPTBYT: Print the byte in Register 'A' on the line printer using the
                ;*        driver routine specified in the line printer DCB at location
                ;*        4025H. DE is lost during execution.
                ;**********
003B            LPTBYT  LD                  ;LD DCB Location 4025H
003E                    JR                  ;JP to OUTBYT
```

```
                ;**********
                ;*BUFFNV: Entry at this location vectors to the buffer input routine at
                ;*       05D9H.  See comments at that location for full details.
                ;**********
0040            BUFFNV  JP

0043                    RET                     ;Not used in Level II
0044                    NOP                     ;Not used in Level II
0045                    NOP                     ;Not used in Level II

                ;**********
                ;*DRIVRV: To save space, all of the above I/O operations jump to the
                ;*       driver address contained in the DCB by way of a JR to this JP
                ;*       Vector to the routine at 03C2H.
                ;**********
0046            DRIVRV  JP

                ;**********
                ;*GETCHR: Scans the keyboard using the KBSCAN routine and waits for a key
                ;*       to be pressed. The character is returned in Reg 'A'. DE lost
                ;*       during call.
                ;**********
0049            GETCHR  CALL                    ;Call KBSCAN Routine
004C                    OR                      ;Set Zero flag if no char
004D                    RET                     ;Return if a key was pressed
004E                    JR                      ;Nothing hit. Try again.

                ;**********
                ;* KBTBL: Keyboard table for use with keyboard driver
                ;*       Special characters table
                ;**********
0050            KBTBL   DEFB    0DH,0DH         ;CR, Shift CR
0052                    DEFB    1FH,1FH         ;CLEAR, Shift CLEAR
0054                    DEFB    01H,01H         ;BREAK, Shift BREAK
0056                    DEFB    5BH,1BH         ;Up Arrow, Shift Up Arrow (ESC)
0058                    DEFB    0AH,1AH         ;Down Arrow (LF), Shift Down Arrow
005A                    DEFB    08H,18H         ;Backspace, CANCEL
005C                    DEFB    09H,19H         ;Tab, 32-Char mode
005E                    DEFB    20H,20H         ;Space, Shift Space

                ;**********
                ;* DELAY: Delay loop. Upon entry, BC loaded with delay count.
                ;*       'A' and 'BC' registers lost. 14.66 msec per loop.
                ;**********
0060            DELAY   DEC                     ;DEC Delay counter
0061                    LD                      ;P/U High order byte of count.
0062                    OR                      ;Determine whether BC=zero
0063                    JR                      ;Loop until counter=0
0065                    RET                     ;Back to caller
```

```
     ;**********
     ;*   NMI: Non-maskable Interrupt Vector
     ;*        Control passes to this point when the RESET button
     ;*        is pressed on the back of the CPU.
     ;**********
0066           NMI     LD                  ;Set dummy stack pointer
0069                   LD                  ;Check if Exp. Interface present
006C                   INC                 ;by getting status of floppy disk
006D                   CP                  ;CP one more than 00+1
006F                   JP                  ;Coldstart if Exp. Int. present
0072                   JP                  ;Warm-start (Restart BASIC without
                                           ;destroying pointers.

     ;**********
     ;*CSTLII: Cold-start for Level II BASIC
     ;*        Routine initializes locations, L3 error vectors,
     ;*        puts returns in disk hook locations,
     ;*        determines memory size, then jumps into BASIC.
     ;**********
0075           CSTLII  LD                  ;Intialize 39 locations
                                           ;  starting at 4080H
0078                   LD                  ;  from data starting at 18F7H
007B                   LD                  ;39 locations loop
007E                   LDIR                ;MOVE!

0080                   LD                  ;Place 3A,00,2C in locations
                                           ;  preceeding I/O buffer
0083                   LD                  ;Store 3A
0085                   INC                 ;Move pointer
0086                   LD                  ;Store 00H
0087                   INC                 ;Move pointer
0088                   LD                  ;Store 2CH
008A                   INC                 ;Move pointer

008B                   LD                  ;Place 41E8 in 40A7H
                                           ;  which points to the
                                           ;  beginning of the I/O buffer.
008E                   LD                  ;Set return vectors for L3 Error
0091                   LD                  ;28 sets of 'JP L3ERR'
0093                   LD                  ;  starting at 4152H
0096                   LD                  ;LD 'JP' instruction
0098                   INC                 ;Move pointer
0099                   LD                  ;Store LSB of L3ERR entry point.
009A                   INC                 ;Move pointer
009B                   LD                  ;Store MSB of L3ERR entry point.
009C                   INC                 ;Move pointer
009D                   DJNZ                ;Loop until 28 sets done
```

```
009F                    LD              ;Puts 21 return statements
00A1                    LD              ;  every third position
00A3                    INC             ;  (since JP'S would be placed
                                        ;   there if the calls
00A4                    INC             ;   are used by DOS, for example)
00A5                    INC             ;Pointer to next entry
00A6                    DJNZ            ;Loop thru 21 sets

00A8                    LD              ;Put a zero at CON0
00AB                    LD              ;  (According to Radio Shack, this
                                        ;   address is always zero).
00AC                    LD              ;LD SP, Low memory stack location
00AF                    CALL            ;Set beginning of string area,
                                        ;  Stack save area.
                                        ;  LD SP with beginning of string
                                        ;  area, devices Reset
00B2                    CALL            ;Clear screen
00B5                    LD              ;Point to "MEMORY SIZE" msg.
00B8                    CALL            ;Print "MEMORY SIZE"
00BB                    CALL            ;Print "? ", Input up to 240 chars.
00BE                    JR              ;If return by <BREAK>, re-enter
00C0                    RST             ;Check buffer for input
00C1                    OR              ;Any non-zero character?
00C2                    JR              ;If a number was input, convert.
                                        ;  else calculate MEMSIZ

    ;**********
    ;*MEMSIZ: Checks each byte in memory for the ability
    ;*        to hold all values. Any error causes termination
    ;*        of the memory test.
    ;**********
00C4            MEMSIZ  LD              ;START AT 434C+1 for mem check
00C7                    INC             ;Skip to next address in RAM
00C8                    LD              ;Check for address 0000H
00C9                    OR              ;  for over-run of counter
00CA                    JR              ;JP if cycle complete
00CC                    LD              ;LD current value from memory
00CD                    LD              ;Save for restoration
00CE                    CPL             ;One's comp. 'A' to check all bits
00CF                    LD              ;Store this new value
00D0                    CP              ;CP stored value with correct value
00D1                    LD              ;Restore original value
00D2                    JR              ;If good CP, test next addr.
00D4                    JR              ;  Else it has found max. mem. size

00D6                    CALL            ;Convert value in buffer
                                        ;  to 2-Byte DE value.
00D9                    OR              ;Test last byte read for 00H
00DA                    JP              ;If not zero, SYNTAX ERROR

00DD                    EX              ;Put value into HL
```

```
00DE                    DEC                 ;Back one location
00DF                    LD                  ;Check ability to hold value
00E1                    LD                  ;Load current value into B
00E2                    LD                  ;LD ADDR,8F(poor value choice)
00E3                    CP                  ;CP with correct value
00E4                    LD                  ;Restore old value
00E5                    JR                  ;JP if Check fails.
                                            ;  Get another MEMSIZ
00E7                    DEC                 ;DEC to proper address
00E8                    LD                  ;LD Minimum mem required
00EB                    RST                 ;CP DE,HL
00EC                    JP                  ;IF HL < DE, Out of Memory Error

00EF                    LD                  ;LD DE,-50 to reserve string space
00F2                    LD                  ;LD Machine size
00F5                    ADD                 ;Sub 50 from HL
00F6                    LD                  ;LD address of string area with
                                            ;  MEMSIZ-50
00F9                    CALL                ;Initialize work area
                                            ;  (same as BCMD 'NEW')
00FC                    LD                  ;Pt to "RADIO SHACK LEVEL II BASIC"
00FF                    CALL                ;Print out msg
0102                    JP                  ;Goto BASIC

0105            DMEMSZ  DEFM    'MEMORY SIZE'
0110                    NOP                 ;Ending delimiter

0111            DRSL2B  DEFM    'RADIO SHACK LEVEL II BASIC'
012B                    DEFB
012C                    NOP                 ;Ending delimiter


        ;**********
        ;* L3ERR: Level 3 error (?L3) for Disk BASIC calls
        ;*        during Level II BASIC
        ;**********
012D            L3ERR   LD                  ;LD L3 error number (2CH)
012F                    JP                  ;Goto error print
```

```
                ;****************************************
                ;*        Graphics Routines             *
                ;****************************************

                ;**********
                ;* POINT: Entry for BASIC command POINT, represented
                ;*        in a BASIC program as a C6H.
                ;**********
0132            POINT   RST                     ;Entry for BCMD POINT (C6)
0133                    XOR                     ;The operation to be performed
                                                ;  depends on the contents of the
                                                ;  'A' register. When A=zero, POINT
0134                    LD                      ;Dummy command. SET enters
                                                ;  at 0135H, in the middle of
                                                ;  this instruction.

                ;**********
                ;*   SET: Entry point for BASIC command SET, represented
                ;*        in a BASIC program as 83H.
                ;**********
0135            SET     LD                      ;Entry for BCMD SET (83).
                                                ;  As above, the determination
                                                ;  between cmds POINT, SET, and
                                                ;  RESET is done thru the 'A' reg.
                                                ;The SET flag is bit 7 (80H)
                                                ;NOTE: When entering from POINT,
                                                ;  this command is invisible.
                                                ;  The opcode is part of LD BC at
                                                ;  location 0134H
0137                    LD                      ;Hide entry point to RESET.

                ;**********
                ;* RESET: Entry point for BASIC command RESET, represented
                ;*        in a BASIC program as 82H.
                ;**********
0138            RESET   LD                      ;Entry for BCMD RESET (82)
                                                ;Also NOTE that this instruction is
                                                ;  invisible when entering
                                                ;  from the above code.
013A                    PUSH                    ;Save the 'A' reg. that denotes the
                                                ;  operation to be performed.
013B                    RST                     ;Find X coord. and return in 'A'.
013C 28                 DEFB    '('             ;  after a syntax check for '('
013D                    CALL
013F                    DEC                     ;DEC parser pointer
0140                    CP                      ;X coord. must be less than 128!
0142                    JP                      ;If not <128, Illegal Function Call
0145                    PUSH                    ;Save X coord.
0146                    RST                     ;Find Y coord. and ret in 'A'
```

```
0147 2C                 DEFB    ','             ;  Syntax check for comma.
0148                    CALL
014B                    CP                      ;Y Coord. must be < 48
014D                    JP                      ;If not <48, Illegal Function Call

     ;**********
     ;* The next section divides the Y coord. by three (3) to get
     ;* the row number <stored in 'D'>, and the remainder <in 'C'>
     ;*     Remember, an integer division (with remainder) is really just
     ;* a series of subtractions, until the value goes below zero and then
     ;* one more increment to bring it positive again.
     ;*     In this case, the number to be divided is in the accumulator, and
     ;* and the quotient is being calculated in reg D.
     ;**********
0150                    LD                      ;Prepare D reg.
0152                    INC                     ;Increment loop
0153                    SUB                     ;Subtract 3 from the accum.
0155                    JR                      ;Has it gone below zero?
0157                    ADD                     ;Restore to positive value
                                                ;  (Get remainder)
0159                    LD                      ;Store remainder in 'C'
015A                    POP                     ;Restore X coord. into 'A'
015B                    ADD                     ;Multiply by two (2)
015C                    LD                      ;Store in 'E'
015D                    LD                      ;This section determines the LSB
015F                    LD                      ;  of the position on the screen,
0160                    RRA                     ;  the value of which is placed in
                                                ;  Register 'E'.
0161                    LD
0162                    LD
0163                    RRA
0164                    LD
0165                    DJNZ
0167                    LD                      ;This section uses the remainder to
0168                    ADC                     ;  to determine the MSB of the
0169                    INC                     ;  byte's location on the screen.
016A                    LD                      ;Value is then placed in 'D'
016B                    XOR                     ;  The location is now in DE.
016C                    SCF
016D                    ADC
016E                    DJNZ
0170                    LD
0171                    LD
0172                    OR
0174                    LD
0175                    LD                      ;LD A, character to be manipulated
0176                    OR
0177                    JP                      ;JP if bit 7 set
                                                ;  <A graphics character>
017A                    LD                      ;Was not graphic.
                                                ;  Set b7. Reset other bits
```

```
017C                    LD
017D                    POP             ;POP type of operation
                                        ;  from PUSH at 013AH
017E                    OR
017F                    LD              ;Restore byte
0180                    JR              ;JP if BCMD 'POINT'
0182                    LD              ;Store byte on screen
0183                    JP              ;JP if BCMD 'SET'
0186                    LD              ;Load bit to reset
0187                    CPL             ;All bits = 1 except bit to reset
0188                    LD
0189                    LD              ;Get character again
018A                    AND             ;Reset bit
018B                    LD              ;Store new value
018C                    RST             ;Clean-up
018D 29                 DEFB    ')'     ;  Syntax chk: closing parenthesis
018E                    RET

018F                    OR              ;SET bit
0190                    JR              ;Finish up.
0192                    AND             ;Check bit for ON/OFF
0193                    ADD             ;If bit ON, RET = -1 (80H).
                                        ;  Else RET=0
0195                    SBC
0196                    PUSH            ;Save HL from destruction
0197                    CALL            ;Routine determines sign of value
019A                    POP             ;Restore HL
019B                    JR

     ;**********
     ;* INKEY: Entry point for BASIC command INKEY$, represented
     ;*        in the BASIC program as C9H
     ;**********
019D            INKEY   RST             ;Entry for BCMD INKEY$ (C9)
019E                    PUSH            ;Save parser pointer
019F                    LD              ;LD last key hit
01A2                    OR
01A3                    JR              ;Skip scan if already have key
01A5                    CALL            ;Scan keyboard for depressed key
01A8                    OR
01A9                    JR              ;Nothing depressed.
                                        ;  Skip next section.
01AB                    PUSH            ;Save char. in 'A'
01AC                    XOR             ;Zero 'A'
01AD                    LD              ;Place in last key hit location
                                        ;  so that this key is not re-read
01B0                    INC
01B1                    CALL            ;40D3=01: 40D4-5 = Dest. for string
01B4                    POP             ;Restore character
01B5                    LD              ;Load string destination
01B8                    LD              ;Put character
```

```
01B9                    JP                  ;Back to caller after character
                                            ;  placed in FPA1 (Vol I)

01BC                    LD                  ;LD pointer to "READY" message
01BF                    LD                  ;Place address in FPA1
01C2                    LD                  ;Define as a string
01C4                    LD                  ;LD TYPFLG for "STRING"
01C7                    POP
01C8                    RET

     ;**********
     ;*   CLS: Entry point used to clear the screen.
     ;*        'A' is lost during execution.
     ;**********
01C9              CLS   LD                  ;Entry BCMD CLS (84)
01CB                    CALL                ;Write Home-cursor to screen
01CE                    LD                  ;LD Clear-to-end char.
01D0                    JP                  ;Write to screen and RET.

     ;**********
     ;* RANDOM: Entry point for BASIC command RANDOM, represented
     ;*         in a BASIC program as 86H.
     ;**********
01D3            RANDOM  LD                  ;Entry for BCMD RANDOM (86)
                                            ;LD A,Memory Refresh Register 'R'
                                            ;  to get a truly "random" number.
01D5                    LD                  ;Store in RNDBYT as part of seed
01D8                    RET                 ;Back to caller

     ;****************************************
     ;*        Cassette I/O Routines         *
     ;****************************************

     ;**********
     ;* CWBIT: Write bit to cassette
     ;**********
01D9             CWBIT  LD                  ;Write bit to cassette
01DC                    CALL                ;Set bit 0 of the
                                            ;    cassette flip flop (CFF) :
                                            ; Reset bit 1
01DF                    LD                  ;Timing delay
01E1                    DJNZ
01E3                    LD
01E6                    CALL                ;Set bit 1 of CFF : Reset bit 0
01E9                    LD                  ;Timing delay
01EB                    DJNZ
01ED                    LD
01F0                    CALL                ;Reset bits 0 & 1 of CFF
01F3                    LD                  ;Timing delay
01F5                    DJNZ
01F7                    RET
```

```
     ;**********
     ;* CTOFF: Turn cassette motor off
     ;**********
01F8            CTOFF   PUSH                    ;Cassette off
01F9                    LD                      ;Clear bit 2 of CFF
01FC                    JR

     ;**********
     ;*  CTON: Turn cassette motor on
     ;**********
01FE             CTON   LD                      ;LD next char (in program usually)
01FF                    SUB                     ;Test for "#" as in "PRINT #-X"
0201                    LD                      ;Default to drive 00
0203                    JR                      ;JP if next char is not "#"
0205                    CALL                    ;Determine the drive number
0208                    RST                     ;  from the PRINT # or INPUT #
                                                ;  statements
0209 2C                 DEFM    ','             ;Syntax check: comma required
020A                    LD
020B                    AND                     ;Check to see Cassette drive number
020C                    ADD                     ;  Is it over 2 ?
020E                    JP                      ;Yes, Illegal Function Call
                                                ;  (No such drive).
0211                    DEC

     ;**********
     ;*DEFDRV: Define cassette drive from Register 'A'.
     ;**********
0212            DEFDRV  LD                      ;Define drive by outputting
                                                ;  to cassette select latch.
0215                    PUSH                    ;Save HL Pointer.
0216                    LD
0219                    CALL                    ;Set bit 2 of CFF
021C                    POP                     ;Restore HL
021D                    RET                     ;Drive Selected

     ;**********
     ;*CLRCFF: Clear Cassette Flip-flop
     ;**********
021E            CLRCFF  LD                      ;Clear CFF

     ;**********
     ;*STATFF: Change status of cassette flip-flop from HL
     ;**********
0221            STATFF  LD                      ;Change status of CFF
0224                    AND                     ;Manipulate old value in CFF
0225                    OR
0226                    OUT                     ;Output to port to change status
0228                    LD                      ;Store new value
022B                    RET                     ;Back to caller
```

```
     ;**********
     ;* CSTAR: Change star in corner of screen during cassette I/O
     ;**********
022C            CSTAR  LD                  ;Change star in corner
                                           ;  at address 3C3FH
022F                   XOR                 ;From <SPACE> to "*" or...
0231                   LD                  ;  from "*" to <SPACE>
0234                   RET                 ;Completed. Return

     ;**********
     ;*CRBYTE: Read single byte from cassette
     ;**********
0235           CRBYTE  PUSH                ;Read byte from cassette
0236                   PUSH
0237                   LD                  ;Read 8 bits
0239                   CALL                ;Read bit from cassette
023C                   DJNZ                ;If not 8 bits, read another
023E                   POP
023F                   POP
0240                   RET                 ;Byte read into 'A', RET

     ;**********
     ;* CRBIT: Read a single bit from the cassette
     ;**********
0241            CRBIT  PUSH                ;Read bit from cassette
0242                   PUSH
0243                   IN                  ;Search for timing bit
0245                   RLA
0246                   JR                  ;Not found. Try again
0248                   LD                  ;Timing delay
024A                   DJNZ
024C                   CALL                ;Clear CFF
024F                   LD                  ;Timing delay
0251                   DJNZ
0253                   IN                  ;Input data bit from cassette port
                                           ;  into high order bit
0255                   LD
0256                   POP                 ;Restore 'A'
0257                   RL                  ;Rotate high order bit into carry
0259                   RLA                 ;Rotate bit into 'A' in low order
025A                   PUSH                ;Save new value
025B                   CALL                ;Clear CFF
025E                   POP                 ;Restore value
0260                   RET

     ;**********
     ;*CW2BYT: Write byte to cassette twice
     ;**********
0261           CW2BYT  CALL                ;Write byte to cassette TWICE!
```

```
     ;**********
     ;* CWBYT: Write byte to cassette
     ;**********
0264            CWBYT  PUSH                ;Write byte to cassette
0265                   PUSH                ;First save registers used
0266                   PUSH
0267                   PUSH
0268                   LD                  ;Eight bits per byte
026A                   LD                  ;Save 'A' contents
026B                   CALL                ;Write timing bit
026E                   LD                  ;Restore character to write
026F                   RLC                 ;Rotate high order bit into CARRY
0270                   LD                  ;Keep this rotated value
                                           ;  for next loop
0271                   JR                  ;If no bit, then delay
0273                   CALL                ;Write data bit
0276                   DEC                 ;DEC bit counter
0277                   JR                  ;Go again if not done
0279                   POP                 ;Restore registers
027A                   POP
027B                   POP
027C                   POP
027D                   RET

027E                   LD                  ;Bit=0. Do NOT write data bit
0280                   DJNZ                ;Delay
0282                   JR                  ;Get another bit

     ;**********
     ;*CTONWL: Cassette ON, Write leader
     ;**********
0284          CTONWL   CALL                ;Cassette on, write leader
0287          CWLDR    LD                  ;Write leader of 255 bytes of 00H
0289                   XOR                 ;Clear 'A' to 00
028A                   CALL                ;Write byte to cassette
028D                   DJNZ                ;Loop through 255 bytes
028F                   LD                  ;LD A, sync. byte (A5H)
0291                   JR                  ;Write sync. byte after leader

     ;**********
     ;*CTONRL: Cassette on, read leader routine
     ;**********
0293          CTONRL   CALL                ;Cassette on
0296                   PUSH                ;Save HL pointer
0297                   XOR                 ;Drive 0
```

```
     ;**********
     ;* CRLDR: Read leader searching for sync. byte
     ;**********
0298           CRLDR  CALL                ;Read leader searching
                                          ;  for sync. byte
029B                  CP                  ;  (A5 is sync byte), by reading
                                          ;  a byte and comparing
029D                  JR                  ;  it to A5H

     ;**********
     ;* CSTARS: Cassette stars placement in corner of screen
     ;**********
029F           CSTARS LD                  ;LD A, '*'
                                          ;Put two stars in corner of screen
02A1                  LD                  ;Put first star at 3C3EH
02A4                  LD                  ;Put second star at 3C3FH
02A7                  POP                 ;Restore HL
02A8                  RET

     ;**********
     ;*SYSTEM: TRS-80 System Tape Utility
     ;*        Reads "SYSTEM" format tapes, and allows
     ;*        transfer to a RAM address either specified by the
     ;*        program tape or by a decimal number entered by the
     ;*        user after a slash [/]
     ;**********
02A9                  CALL                ;Get transfer address for SYSTEM
                                          ;  from tape.
02AC                  LD                  ;Load into transfer addr. location
02AF                  CALL                ;Cassette off

02B2           SYSTEM CALL                ;System entry point (BCMD AE)
02B5                  LD
02B8                  CALL                ;Output CR to current device
02BB                  LD                  ;LD A,'*'
02BD                  CALL                ;Output '*' (Display user prompt)
02C0                  CALL                ;Input buffer of 240 characters
                                          ;  after "? " prompt
02C3                  JP                  ;If ended on <BREAK> GOTO BASIC
02C6                  RST                 ;Test buffer
02C7                  JP                  ;If nothing there, SYNTAX ERROR
02CA                  CP                  ;CP '/'
02CC                  JR                  ;JP if match to "SYSGO"
02CE                  CALL                ;Cassette on, find sync byte,
                                          ;  put stars in corner
```

```
02D1                CALL            ;Read byte from cassette
02D4                CP              ;Search for byte preceding title
02D6                JR
02D8                LD              ;Load max number of Chars. in title
02DA                LD              ;LD first char. in title into 'A'
02DB                OR              ;Check for end-of-buffer
02DC                JR              ;JP if End-of-Title found
02DE                CALL            ;Read byte from cassette
02E1                CP              ;CP byte from cassette
                                    ; with next byte in title
02E2                JR              ;Get another byte if no match
02E4                INC             ;INC buffer pointer

                ;Note:   In some machines, the INC HL comes before
                ;        the JP NZ. If this is the case in a machine,
                ;        whenever the bytes do not match, SYSTEM will never
                ;        find the correct title, since the buffer pointer
                ;        is never reset!

02E5                DJNZ            ;Go back and get another char.
02E7                CALL            ;Change star
02EA                CALL            ;Read byte from cassette
02ED                CP              ;CP BYTE,78H which is the byte
                                    ;  preceeding the transfer addr.
                                    ;  at the end of the tape.
02EF                JR              ;JP if match to READ ADDR, CTOFF
02F1                CP              ;CP BYTE,3CH which is the byte
                                    ;  preceeding the load address
02F3                JR              ;JP If no match.

                ;Note:   Each separate record on a system tape must have
                ;        a load address preceeding it.
                ;        At this point, the record separator has been located.
                ;        The number of bytes in the record, the load address,
                ;        the data record, and the checksum (at the end) will be
                ;        read.
                ;        The load address is INCLUDED in the checksum!

02F5                CALL            ;Read number of bytes in record
02F8                LD              ;Store in B
02F9                CALL            ;Get load address
02FC                ADD             ;Add load address to checksum, too.
02FD                LD              ;Save checksum in 'C'.
02FE                CALL            ;Read data byte
0301                LD              ;Save at proper address
0302                INC             ;INC load address
0303                ADD             ;Add previous checksum
0304                LD              ;Save new checksum
0305                DJNZ            ;Get another byte in record
```

```
0307                    CALL            ;Get the recorded checksum
                                        ;  at the end-of-rec on tape
030A                    CP              ;CP with computed checksum
030B                    JR              ;If OK, get another record
030D                    LD              ;Checksum error
030F                    LD              ;Put 'C' in corner replacing '*'
0312                    JR              ;Get another record
0314            GETADR  CALL            ;Get an address from tape
0317                    LD              ;LD L, LSB
0318                    CALL            ;Get next byte
031B                    LD              ;LD H, MSB
031C                    RET             ;Done
031D                    EX              ;Goto address, either from
                                        ;  /<Decimal #>
                                        ;  or from address from tape
031E                    LD              ;LD system transfer address
0321                    EX              ;Put in DE
0322                    RST             ;Check input buffer
0323                    CALL            ;If a number present, convert to
                                        ;  a two-byte DE value
0326                    JR              ;Back to system if no convert
0328                    EX              ;Switch address to HL for JP
0329                    JP              ;GO!!!

    ;**********
    ;*DSPCHR: Display the byte in 'A' on the current device.
    ;*       The current device is determined by the flag at 409CH.
    ;*       If the flag has bit 7 set, output to the cassette.
    ;*       If other than zero (00H), send to line printer.
    ;*       If zero, send to the video monitor.
    ;*       DE not destroyed.
    ;**********

032A            DSPCHR  PUSH
032B                    LD              ;Save byte to output in 'C'
032C                    CALL            ;Disk hook to 41C1H
032F                    LD              ;LD I/O flag
0332                    OR              ;Set flags depending on value
0333                    LD              ;Restore output character
0334                    POP             ;Restore BC
0335                    JP              ;If bit 7 set, output to cassette
0338                    JR              ;Output to printer if non-zero
033A                    PUSH
033B                    CALL            ;Output character to monitor
033E                    PUSH
033F                    CALL            ;Call POSIND
                                        ;  (line position indicator)
0342                    LD              ;Save position in CRTPOS
0345                    POP             ;Restore output byte
0346                    POP
0347                    RET
```

```
     ;**********
     ;*POSIND: Determines the cursor position on the screen
     ;*        taking into consideration 32-char mode.
     ;**********
0348                 LD                  ;Check bit 3 of cassette status
                                         ;  byte for 32-char mode
034B                 AND
034D                 LD                  ;LD A,Cursor position (OLD)
0350                 JR                  ;JP if double width
0352                 RRCA                ;Divide by two
0353                 AND                 ;Make sure value < 32
0355                 AND                 ;Make sure value < 64
0357                 RET                 ;Completed

     ;**********
     ;*KBDSCN: Keyboard scan, saving DE register pair.
     ;*        Scans the keyboard for input using the
     ;*        routine starting at 2BH, but does not destroy DE.
     ;**********
0358          KBDSCN CALL                ;Disk hook to 41C4H
035B                 PUSH                ;Save DE
035C                 CALL                ;Scan keyboard.
                                         ;  Return char. in 'A'
035F                 POP                 ;Restore DE
0360                 RET

     ;**********
     ;*INCHRS: Inputs up to 240 characters using the BUFFIN
     ;*        Routine at 05D9H.
     ;*        Exit: HL points to beginning of buffer-1
     ;*          BC Saved, 'A' zero if no <BREAK>
     ;*          Carry if <BREAK>, end of buffer has 00H instead of 0DH
     ;**********
0361          INCHRS XOR                 ;Clear 'A' for next two steps
0362                 LD                  ;Zero INKEY$ byte
0365                 LD                  ;LD current line position on video
                                         ;  with zero (00H), even though it
                                         ;  might be in another position.
0368                 CALL                ;Disk hook to 41AFH
036B                 PUSH                ;Save BC
036C                 LD                  ;LD HL, Beginning of input buffer
036F                 LD                  ;Set up for input of 240 chars.
0371                 CALL                ;Call buffer-input routine
0374                 PUSH                ;Save flags
0375                 LD                  ;Add number of chars. to beginning
                                         ;  of buffer
0376                 LD                  ;Zero MSB for ADD
0378                 ADD                 ;Point HL to end of buffer
0379                 LD                  ;Put a zero, replacing CR
037B                 LD                  ;Point HL to beginning of buffer
```

```
037E                    POP             ;Restore flags to check for <BREAK>
037F                    POP
0380                    DEC             ;Point HL to beginning of buffer-1
0381                    RET             ;Return if BUFFIN ended on <BREAK>
0382                    XOR             ;Clear 'A'
0383                    RET
     ;**********
     ;*GTDCHR: Scan keyboard using KBDSCN at 0358H
     ;*       and wait for input. DE not destroyed
     ;**********
0384           GTDCHR   CALL            ;Scan keyboard, saving DE
0387                    OR              ;Check for input (NZ)
0388                    RET             ;Return if character present
0389                    JR              ;Scan again
     ;**********
     ;*RSTDEV: Resets current device to video monitor.
     ;*       Checks line printer and forces CR if in middle of line.
     ;**********
038B           RSTDEV   XOR             ;Clear 'A'
038C                    LD              ;Set current device
                                        ;  to video display
038F                    LD              ;Check for characters in
                                        ;  line printer buffer
0392                    OR
0393                    RET             ;Nothing there. No need to finish
0394                    LD              ;Force a CR
0396                    PUSH            ;Save DE
0397                    CALL            ;Output CR to printer
                                        ;  (clears printer buffer)
039A                    POP             ;Restore DE
039B                    RET
     ;**********
     ;*LPDCHR: Output byte in 'A' to line printer
     ;*       Registers saved
     ;**********
039C           LPDCHR   PUSH            ;Save character to print
039D                    PUSH            ;Save registers
039E                    PUSH
039F                    LD              ;Save byte to print
03A0                    LD              ;Initialize line position
                                        ;  at zero (00H)
03A2                    CP              ;CP char., form feed
03A4                    JR              ;JP if form feed
03A6                    CP              ;CP char., linefeed
03A8                    JR              ;JP if not linefeed
03AA                    LD              ;Change LF to CR
03AC                    LD              ;Store in 'C'
03AD                    CP              ;CP char., CR
03AF                    JR              ;JP if CR
03B1                    LD              ;LD A, Current line Position
03B4                    INC             ;INC line position
```

```
03B5                    LD                  ;Store in 'E'
03B6                    LD                  ;Store current line position in 'A'
03B7                    LD                  ;  then store it in LPTPOS
03BA                    LD                  ;Restore char. to print from 'C'
03BB                    CALL                ;Output character to printer
03BE                    POP                 ;Restore registers
03BF                    POP
03C0                    POP
03C1                    RET
     ;**********
     ;*DRIVER: I/O Driver, using the Device Control Block (DCB)
     ;*         Entry:  LD DE, DCB location
     ;*                 LD A, Character
     ;*                 LD B, type of operation
     ;*                 BC pushed
     ;*         Exit:   Branch to driver address at DCB+1,DCB+2
     ;*                 Returns to driver to restore registers
     ;**********
03C2           DRIVER   PUSH                ;Save registers
03C3                    PUSH
03C5                    PUSH                ;LD IX,DE
03C6                    POP
03C8                    PUSH
03C9                    LD                  ;Set up return address
03CC                    PUSH                ;Push address
03CD                    LD                  ;Save character to print
03CE                    LD                  ;LD DCB type
03CF                    AND                 ;AND with type of operation
03D0                    CP                  ;CP with type of operation
03D1                    JP                  ;JP to "Driver Call Illogical"
                                            ;  if not the same.
                                            ;  (EX: Input from a CRT invalid)
03D4                    CP                  ;CP with type of operation
                                            ;  denoting output
03D6                    LD                  ;LD LSB of driver address
03D9                    LD                  ;LD MSB of driver address
03DC                    JP                  ;GOTO driver for device
03DD           DRVRET   POP                 ;After return from driver,
                                            ;  restore registers
03DE                    POP
03E0                    POP
03E1                    POP
03E2                    RET

     ;**********
     ;* KEYIN: Scans the keyboard, searching for a newly
     ;*        depressed key.  The result of the scan
     ;*        is returned in resister 'A'.  This is the
     ;*        routine called by the DRIVER, as specified
     ;*        the DCB.
     ;**********
```

```
03E3            KEYIN   LD              ;LD Keyboard image start.
                        ;  The keyboard image table
                        ;  contains the value of the last
                        ;  scan of the keyboard for each of
                        ;  the seven locations
03E6                    LD              ;LD the start of the keyboard
                        ;  BC now contains the first
                        ;  address of the locations
                        ;  to be scanned
03E9                    LD              ;Zero key counter
03EB                    LD              ;Load first character from keyboard
03EC                    LD              ;Save result
03ED                    XOR             ;XOR with old value
03EE                    LD              ;Store this new value in
                        ;  keyboard image
03EF                    AND             ;Check to see if the key
                        ;  was pressed before
03F0                    JR              ;JP if new key depressed
03F2                    INC             ;INC key counter
03F3                    INC             ;INC keyboard image location.
03F4                    RLC             ;GOTO next keyboard position
03F6                    JP              ;IF not 7 locations scanned,
                        ;  go again
03F9                    RET             ;No key was found
03FA                    LD              ;Save the "LIT" bit
03FB                    LD              ;Determine 8 * ROW#
03FC                    RLC             ;*2
03FD                    RLC             ;*4
03FE                    RLC             ;*8
03FF                    LD              ;Save 8 * ROW#
0400                    LD

    ;**********
    ;*  The next section adds to 8 * ROW# the column number
    ;*  (i.e., if bit 3 is on, then the key pressed was in the
    ;*  3rd column, so three (3) is added to register 'D'.
    ;*  Remember, bit 0 denotes column 0.)
    ;**********

0402                    LD              ;LD C (This value will have only
                        ;  one bit on.)
0403                    AND             ;Check if the bits match up
0404                    JR              ;JP if bits match
0406                    INC             ;INC value
0407                    RLC             ;Shift comparison bit
0409                    JR              ;Test next bit
040B                    LD              ;Test for <SHIFT>
040E                    LD              ;Put value of <SHIFT> in 'B'
040F                    LD              ;Load the semi-converted character
0410                    ADD             ;If alphabetic, convert to
                        ;  correct value
```

```
0412            CP
0414            JR              ;JP if non-alphabetic
0416            RRC             ;Test for <SHIFT>
0418            JR              ;JP if no <SHIFT>
041A            ADD             ;Convert to lowercase
041C            LD              ;Save value
041D            LD              ;Check for down arrow depressed
0420            AND             ;  (Shift-downarrow = Control)
0422            JR              ;Jump if not a control character
0424            LD              ;Restore char.
0425            SUB             ;Convert to control code
0427            JR              ;Skip next section
0429            SUB             ;SUB 70H for non-alphabetic
042B            JR              ;JP if a special character
042D            ADD             ;Convert to Numeric/Symbol
042F            CP              ;Manipulate value to get
                                ;  proper code
0431            JR
0433            XOR
0435            RRC             ;Check for <SHIFT>
0437            JR              ;JP if NO <SHIFT>
0439            XOR             ;Adjust for <SHIFT>
043B            JR              ;DONE
043D            RLC             ;Table look-up of special chars.
043E            RRC             ;Check <SHIFT>
0440            JR              ;JP if no <SHIFT>
0442            INC             ;Shist chars. in table are located
                                ;  one location after un-shifted
0443            LD              ;LD HL, Beginning
                                ;  of special char. Table
0446            LD              ;Find location of char. in table
0447            LD
0449            ADD
044A            LD              ;LD char. from table
044B            LD              ;Scan complete.
                                ;  Save char. for delay
044C            LD              ;Delay loop
044F            CALL            ;Delay
0452            LD              ;Restore character
0453            CP              ;Check for <BREAK>
0455            RET             ;NO <BREAK>
0456            RST             ;BREAK! RST 40 (Debug)
0457            RET             ;Back to caller
```

```
     ;****************************************
     ;*        Video Display Driver          *
     ;*        Entry:   IX = DCB location    *
     ;*                 'C' = Character      *
     ;*                       to display     *
     ;****************************************
0458           VIDEO  LD                    ;LD HL, Cursor position
045B                  LD
045E                  JR                    ;Jump if this is input request
0460                  LD                    ;LD A, character at cursor
0463                  OR                    ;Anything there?
0464                  JR                    ;JP if nothing at cursor
0466                  LD                    ;Restore character at cursor
                                            ;  on display
0467                  LD                    ;Restore character to print
0468                  CP                    ;CP <SPACE>
046A                  JP                    ;JP if a control code
                                            ;  (Special Character Routines)
046D                  CP
046F                  JR                    ;JP if a graphic
                                            ;  or space compression code
0471                  CP
0473                  JR                    ;JP if not alphabetic
0475                  SUB
0477                  CP
0479                  JR                    ;JP if uppercase
047B                  SUB                   ;Convert to uppercase
047D                  CALL                  ;Write character
0480                  LD                    ;Determine MSB of cursor location
0481                  AND
0483                  OR
0485                  LD
0486                  LD                    ;Save character at cursor position
0487                  LD                    ;LD A, character at cursor
048A                  OR                    ;Anything there?
048B                  JR                    ;JP if nothing there
048D                  LD                    ;Save character at cursor
0490                  LD                    ;Write cursor
0492                  LD                    ;Save new cursor position
0495                  LD
0498                  LD                    ;Restore character to print
0499                  RET                   ;Back to caller
                                            ;  (Usually the DRIVER)

049A                  LD                    ;LD A, Character at cursor
049D                  OR
049E                  RET                   ;Return if something there
049F                  LD                    ;LD A, Character on screen
                                            ;  at cursor position
04A0                  RET
```

```
04A1                LD              ;Return cursor position
                                    ;  to beginning of line
04A2                AND
04A4                LD
04A5                RET

04A6                CP
04A8                JR              ;JP if a graphic character
04AA                SUB             ;Convert to 00-63 spaces
04AC                JR              ;IF 00, already finished
04AE                LD              ;Set up loop count
04AF                LD              ;Will be sending <SPACE>'s
04B1                CALL            ;Output <SPACE>
04B4                DJNZ            ;Loop if not finished
04B6                JR              ;Done sending <SPACE>'s

04B8                LD              ;Turn cursor on
04B9                LD
04BC                RET

04BD                XOR             ;Turn cursor off
04BE                JR
04C0                LD              ;Home cursor to first position
                                    ;  on screen
04C3                LD              ;Load cassette status byte
                                    ;  for 32-char. mode
04C6                AND             ;32-Char off!
04C8                LD              ;Restore cassette status byte
04CB                OUT             ;Output lower 4-bits
                                    ;  (turn off 32-char.)
04CD                RET

04CE                DEC             ;Backspace and erase previous char.
04CF                LD              ;Check for 32-char mode
04D2                AND
04D4                JR              ;JP if 64-char mode
04D6                DEC             ;Backspace twice for 32-char
04D7                LD              ;Put a <SPACE>
04D9                RET
04DA                LD              ;Backspace cursor
04DD                AND             ;Check for 32-char mode
04DF                CALL            ;Backspace twice when in 32-char
04E2                LD
04E3                AND             ;Set flags for beginning of line
04E5                DEC             ;Backspace cursor
04E6                RET             ;Return if not at beginning of line

04E7                LD              ;Downward linefeed
04EA                ADD             ;Move cursor to next line
04EB                RET
```

```
04EC            INC             ;Advance cursor
04ED            LD
04EE            AND             ;Check for beginning of line
04F0            RET             ;Return if not

04F1            LD              ;Upward linefeed (LD -40H)
04F4            ADD             ;Subtract one line from current pos
04F5            RET

04F6            LD              ;Set 32-char. mode
04F9            OR              ;Set 32-char bit
04FB            LD              ;Store in cassette status byte
04FE            OUT             ;Turn on 32-char. mdoe
0500            INC             ;Correct cursor position
                                ;  for 32-char. mode.
0501            LD
0502            AND
0504            LD
0505            RET

0506            LD              ;JP from above for control chars.
0509            PUSH            ;Set up return address (0480H)
050A            CP              ;<BACKSPACE>?
050C            JR              ;<BACKSPACE>!
050E            CP              ;<LINEFEED>?
0510            RET             ;Ret if value < 0AH
0511            CP              ;Check for linefeed.
0513            JR              ;JP <LINEFEED CHARACTER>
0515            JR              ;JP if <CURSOR ON>
0517            CP
0519            JR              ;JP if <CURSOR OFF>
051B            CP
051D            JR              ;JP if <32-CHAR MODE>
051F            CP
0521            JR              ;JP if <BACKSPACE>
0523            CP
0525            JR              ;JP if <ADVANCE CURSOR>
0527            CP
0529            JR              ;JP if <DOWN LINEFEED>
052B            CP
052D            JR              ;JP if <UP LINEFEED>
052F            CP
0531            JR              ;JP if <HOME CURSOR>
0533            CP
0535            JP              ;JP if <CURSOR - BEGINNING OF LINE>
0538            CP
053A            JR              ;JP if <ERASE TO END-OF-LINE>
053C            CP
053E            JR              ;JP if <CLEAR END-OF-SCREEN>
0540            RET
```

```
0541            LD              ;Write character onto screen
0542            INC             ;INC pointer
0543            LD              ;Check for 32-char mode
0546            AND
0548            JR              ;JP if 64-CHAR
054A            INC             ;INC again for double width
054B            LD              ;LD MSB of cursor position
054C            CP              ;CP with highest allowable+1
054E            RET             ;Return if not off screen
054F            LD              ;DEC cursor position one line
0552            ADD             ;  By adding -64 to cursor position
0553            PUSH            ;SAVE cursor position
0554            LD              ;Scroll screen
                                ;  (LD DE, First char. loc.)
0557            LD              ;LD HL, First character location
                                ;  in second line
055A            PUSH            ;SAVE BC
055B            LD              ;Scroll full page
055E            LDIR            ;Move characters!
0560            POP
0561            EX              ;LD HL, Beginning of last line
0562            JR              ;GOTO CLEAR-TO-END-OF-LINE
0564            LD              ;Entry for <CR>'S
0565            AND             ;GOTO Beginning-of-line
0567            LD
0568            PUSH            ;Save pointer
0569            LD              ;LD DE, 64
                                ;  to get pointer to next line
056C            ADD             ;Add one line to cursor position
056D            LD
056E            CP              ;Over end?
0570            JR              ;JP if over end-of-screen
0572            POP             ;POP Position
0573            PUSH            ;Erase-to-end-of-line
0574            LD              ;Set pointers
0575            LD
0576            OR
0578            LD
0579            INC
057A            JR              ;Clear to end of line

057C            PUSH            ;Clear to end-of-screen
057D            LD              ;Clear line to blanks
0580            LD              ;LD Cursor positonN, <SPACE>
0582            INC             ;INC pointer
0583            LD
0584            CP
0585            JR              ;JP if not done
0587            LD
0588            CP
0589            JR              ;JP if not done
```

```
058B                    POP                     ;Pop pointer
058C                    RET

        ;****************************************
        ;*        Line Printer Driver           *
        ;*        Entry:  IX = DCB Location     *
        ;*                'C' = Character       *
        ;*                      to be printed   *
        ;*                                      *
        ;****************************************
058D            LPTDRV  LD                      ;Restore char. to be printed from C
058E                    OR                      ;Skip nulls. Returns status
058F                    JR                      ;Return status if NULL
0591                    CP                      ;CP VT (Type of linefeed)
0593                    JR                      ;JP if match
0595                    CP                      ;CP Formfeed
0597                    JR                      ;JP if no match
0599                    XOR                     ;Clear 'A'
059A                    OR                      ;See if lines/page is set
059D                    JR                      ;JP if not set
059F                    LD                      ;LD number of lines/page
05A2                    SUB                     ;Subtract value in page counter
05A5                    LD                      ;Store in 'B' (count)
05A6                    CALL                    ;Check if printer is ready
05A9                    JR                      ;Loop if not
05AB                    LD                      ;LD A, linefeed
05AD                    LD                      ;Output linefeed to printer
05B0                    DJNZ                    ;Output "B" linefeeds to printer
05B2                    JR                      ;Set line counter to zero, RET
05B4                    PUSH                    ;Output character to printer
05B5                    CALL                    ;Printer ready?
05B8                    JR                      ;JP if busy.
05BA                    POP                     ;Restore character to print
05BB                    LD                      ;Output character to printer
05BE                    CP                      ;Is it a <CR>?
05C0                    RET                     ;Return if not a <CR>
05C1                    INC                     ;INC line counter
05C4                    LD                      ;Load line counter
05C7                    CP                      ;Beyond end of paper?
05CA                    LD                      ;Restore printed character
05CB                    RET                     ;Return if not to end
05CC                    LD                      ;Zero line counter
05D0                    RET

        ;**********
        ;*PSTATU: Check status of printer
        ;**********
05D1            PSTATU  LD                      ;Check status of printer
05D4                    AND
05D6                    CP                      ;If ready, Z Flag set
05D8                    RET
```

```
                ;**********
                ;*BUFFIN: Buffer input routine
                ;*       Inputs a maximum of "B" characters into a buffer
                ;*       pointed to by HL on entry.
                ;*       A RETURN is executed either after an <ENTER>
                ;*       or after <BREAK>. If by a <BREAK>, the carry is set.
                ;*
                ;*       During execution:
                ;*        DE lost. HL points to buffer. BC used for number of chars.
                ;*        'A' stores chars.
                ;*
                ;*       Upon completion:
                ;*        HL points to the beginning of buffer,
                ;*        Carry flag set if ended with <BREAK>,
                ;*        'B' holds the number of characters in buffer
                ;**********
05D9            BUFFIN  PUSH            ;Save buffer pointer for return
05DA                    LD              ;Cursor on for input
05DC                    CALL            ;Output cursor on character
05DF                    LD              ;LD 'C' with number of chars.
                                        ;  allowed. 'B' will be decremented
05E0                    CALL            ;Scan keyboard and wait for KEY
05E3                    CP              ;Check for control code
05E5                    JR              ;JP if not control
05E7                    CP              ;Check for <ENTER>
05E9                    JP              ;JP if <ENTER>
05EC                    CP              ;Check for <CLEAR>
05EE                    JR              ;JP if <CLEAR>
05F0                    CP              ;Check for <BREAK>
05F2                    JR              ;JP if <BREAK>
05F4                    LD              ;Load return address
05F7                    PUSH            ;Push return address onto stack
05F8                    CP              ;Check for <BACKSPACE>
05FA                    JR              ;JP if <BACKSPACE>
05FC                    CP              ;Check for <CANCEL>
05FE                    JR              ;JP if <CANCEL> (Shift BACKSPACE)
0600                    CP              ;Check for <TAB>
0602                    JR              ;JP if <TAB>
0604                    CP              ;Check for <32-Char. Mode>
0606                    JR              ;JP if <32-char. mode> (Shift TAB)
0608                    CP              ;Check for <LINEFEED>
060A                    RET             ;Return to 05E0
                                        ;  if illegal control code
060B                    POP             ;Take 05E0 address off stack
060C                    LD              ;Put character in buffer
060D                    LD              ;LD A, number of characters
                                        ;  left in buffer
060E                    OR              ;Compare against itself
                                        ;  (only time zero flag set,
```

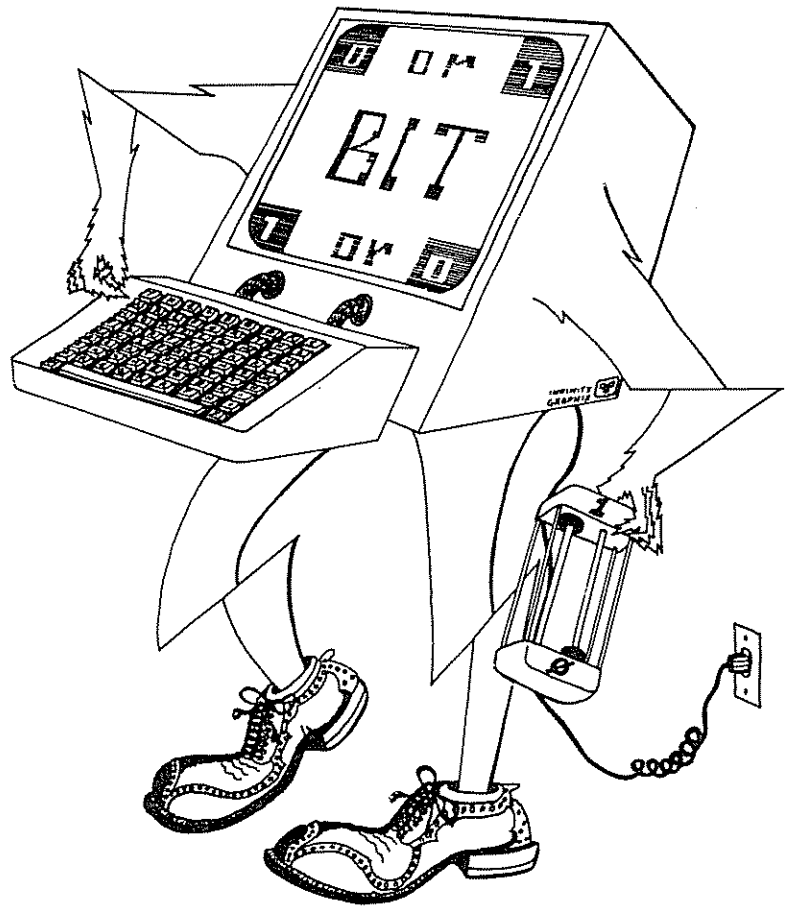
```
                                        ;    is when accumuator = zero)
060F                JR                  ;No more room left.
                                        ;  Wait for either <ENTER> or <BRK>
                                        ;  before returning.
0611                LD                  ;Restore character for output
0613                CALL                ;Output character
0616                DEC                 ;DEC # of chars. left in buffer.
0617                JR                  ;Get another character

    ;**********
    ;* Hitting the <CLEAR> key clears screen and
    ;* resets pointers to orginal state
    ;**********
0619                CALL                ;Clear screen
061C                LD                  ;Restore # chars. left in buffer
                                        ;  to original value
061D                POP                 ;Restore buffer pointer
061E                PUSH                ;PUSH back on stack buffer pointer
                                        ;  for possible re-use
061F                JP                  ;Back to the beginning!

    ;**********
    ;* Shift backspace clears one character at a time until the
    ;* beginning of the line, or until a LF is reached in the buffer.
    ;**********
0622                CALL                ;Clear one character at a time
0625                DEC                 ;DEC buffer pointer after BKSP
0626                LD                  ;Check char. at pointer
0627                INC                 ;INC pointer before compare
0628                CP                  ;CP with LF
062A                RET                 ;RET. if LF since
                                        ;  can't BKSP over LF
062B                LD                  ;Entry from shift BKSP.
                                        ;  Checks for beginning of buffer.
062C                CP                  ;Is B=number chars. allowed?
062D                JR                  ;JP if still have to backspace
062F                RET                 ;Shift BKSP complete

    ;**********
    ;* BACKSPACE goes back one char. unless already at beginning of buffer
    ;* or at a LF.
    ;**********
0630                LD                  ;Check for beginning of buffer
0631                CP                  ;Is B=number chars. allowed?
0632                RET                 ;Already at beginning of buffer
0633                DEC                 ;DEC pointer in buffer
0634                LD                  ;Check character for LF
0635                CP                  ;CP with Linefeed character
0637                INC                 ;INC Pointer
0638                RET                 ;Char. a LF, so RETURN.
0639                DEC                 ;DEC pointer after INC at 0637H.
```

```
063A                    LD                  ;Load backspace char. for output
063C                    CALL                ;Output backspace
                                            ;  and erase previous character
063F                    INC                 ;One more character allowed
                                            ;  in buffer after BACKSPACE
0640                    RET                 ;BACKSPACE completed

    ;**********
    ;* 32-character mode turns display to 32-char. mode
    ;* but does not store the character (17H) in buffer.
    ;**********
0641                    LD                  ;LD character for
                                            ;  double width chars.
0643                    JP                  ;Output this char.
                                            ;  Return to 05E0 (From PUSH)

    ;**********
    ;* TAB advances cursor and buffer pointer to next TAB position.
    ;**********
0646                    CALL                ;Determine position on screen
                                            ;  (Returns in 'A')
0649                    AND                 ;Determine # of chars.
                                            ;  until next stop
064B                    CPL
064C                    INC
064D                    ADD
064F                    LD                  ;Save # chars. in 'E'
0650                    LD                  ;Check if space in buff.
0651                    OR
0652                    RET                 ;RET. if no space left
0653                    LD                  ;LD A,<SPACE>
0655                    LD                  ;Put <SPACE> in buffer
0656                    INC                 ;INC buff. pointer
0657                    PUSH                ;Save DE since output
                                            ;  will kill it.
0658                    CALL                ;Output space
065B                    POP                 ;Restore DE
065C                    DEC                 ;DEC space left in buffer.
065D                    DEC                 ;DEC # <SPACE>'S till next TAB pos.
065E                    RET                 ;RETURN if finished
065F                    JR                  ;JP back and check
                                            ;  for space in buffer.

    ;**********
    ;* BREAK sets carry and places CR (0D) at buffer pointer loc.
    ;**********
0661                    SCF                 ;Set carry flag for break detect
                                            ;<ENTER> BEGINS HERE ALSO.
0662                    PUSH                ;Save flags
0663                    LD                  ;LD <ENTER> character
0665                    LD                  ;Place in buffer
```

```
0666                    CALL            ;Output <ENTER>
0669                    LD              ;Cursor OFF
066B                    CALL            ;Output Cursor Off
066E                    LD              ;Determine number of chars.
                                        ;  in buffer
066F                    SUB             ;Subtract current left from max.
0670                    LD              ;Return in B
0671                    POP             ;Restore flags
0672                    POP             ;Return with HL pointing
                                        ;  to beginning of buffer.
0673                    RET             ;DONE!

     ;**********
     ;*COLDST: Coldstart Routine
     ;**********
0674          COLDST    OUT             ;Cassette off
0676                    LD              ;LD vectors and DCBS to memory
0679                    LD              ;  Starting at 4000H
067C                    LD              ;  From 06D2H for 36H locations
067F                    LDIR            ;Move!
0681                    DEC
0682                    DEC
0683                    JR              ;Repeat this procedure
                                        ;  while RAM warms up
0685                    LD              ;Zero next 39 bytes of RAM
0687                    LD              ;Store zero
0688                    INC             ;INC RAM Pointer
0689                    DJNZ            ;Loop through 39 locations
068B                    LD              ;Check for <BREAK>
068E                    AND             ;  at bit 2
0690                    JP              ;JP to LEVEL II Coldstart
                                        ;  if <BREAK> pressed
0693                    LD              ;LD Disk boot stack pointer
0696                    LD              ;Check for expansion interface
                                        ;  and disks
0699                    INC             ;INC Floppy Disk Controller Status
069A                    CP              ;A status of 00 or FF is bad
069C                    JP              ;JP to LVLII Coldstart
                                        ;  if missing or busy
     ;**********
     ;*DISKBT: Disk boot strap
     ;**********
069F          DISKBT    LD              ;Disk bootstrap loads disk
                                        ;  operating system by first
                                        ;  loading bootstrap on track 00,
                                        ;  sector 00 on drive 00.
06A1                    LD              ;Start drive 0
06A4                    LD              ;LD HL, Disk controller address
06A7                    LD              ;LD DE, Disk data register address
06AA                    LD              ;Restore head to track 0
06AC                    LD              ;Delay during head movement
```

```
06AF                    CALL                ;DELAY Call
06B2                    BIT                 ;Check FDC Status
06B4                    JR                  ;Loop if not ready yet
06B6                    XOR                 ;Clear A
06B7                    LD                  ;Load sector register with zero (0)
06BA                    LD                  ;LD BC, Destination of loader
                                            ;  program (4200H)
06BD                    LD                  ;LD Read sector command
06BF                    LD                  ;Read sector zero
06C0                    BIT                 ;Byte ready to be read?
06C2                    JR                  ;Loop if no byte ready
06C4                    LD                  ;LD byte from disk
06C5                    LD                  ;Store in RAM
06C6                    INC                 ;INC Pointer
06C7                    JR                  ;JP if not a whole sector loaded
06C9                    JP                  ;Goto loader!

     ;**********
     ;* BASIC: This is the proper entry to BASIC to
     ;*        avoid an error
     ;**********
06CC            BASIC   LD                  ;Proper re-entry to BASIC
06CF                    JP                  ;Exit

     ;**********
     ;* The rest of the bootstrap section of the ROM from 06D2-0707H
     ;* is data loaded into RAM by the COLDSTART routine
     ;**********
06D2            RSTRTS  JP                  ;RST'S Loaded into RAM @ 4000H
06D5                    JP
06D8                    JP
06DB                    JP
06DE                    RET
06DF 0000               DEFW    0000H
06E1                    RET
06E2 0000               DEFW    0000H
06E4                    EI                  ;Enable Interrupts
06E5                    RET

06E6 00                 DEFB    00H
06E7 01                 DEFB    01H         ;Keyboard device type
06E8 E303               DEFW    03E3H       ;Keyboard driver address
06EA 0000               DEFW    0000H
06EC 00                 DEFB    00H
06ED 4B49               DEFM    'KI'        ;Keyboard device name

06EF 07                 DEFB    07H         ;Display device type
06F0 5804               DEFW    0458H       ;Display driver address
06F2 003C               DEFW    3C00H       ;Cursor position
06F4 00                 DEFB    00H         ;Character at cursor position
06F5 444F               DEFM    'DO'        ;Display device name
```

```
06F7 06                 DEFB    06H             ;Line printer device type
06F8 8D05               DEFW    058DH           ;Printer driver address
06FA 43                 DEFB    43H             ;Lines per page + 1 (default 67)
06FB 00                 DEFB    00H             ;Line counter
06FC 00                 DEFB    00H
06FD 5052               DEFM    'PR'            ;Printer device name

06FF                    JP
0702                    RST
0703                    NOP
0704                    NOP
0705                    LD
0707                    RET
0707            ENDDAT  EQU                     ;END OF DATA


0703                    NOP
0704                    NOP
0705                    LD                      ;Clear 'A'
0707                    RET
```



Other I/O Routines

There are a few routines in the ROM which may be useful to the assembly language programmer. They are outlined here since they do not really fit in any of the chapters describing the individual units.

Did you ever need to get access to the **Program Counter** (PC) to figure out where the current program is executing in memory? The following routine loads 'HL' with the contents of the updated PC:

```
WHERE   CALL  000BH          ;LD HL,PC
THISAD  EQU   $              ;HL will contain the
                             ; address of THISAD
                             ; in memory
```

Parsing through a buffer is facilitated by one of the RSTs. RST 16 is used by the ROM to process BASIC programs in memory, but we can use it to scan a buffer of our own. Simply point 'HL' to the address one location before the position of the string and do an RST 16. The routine increments 'HL' and loads the character into 'A'. The flags are set to denote the type of character; if the Z flag is set, the character in 'A' is either a 00H or a colon (:). The Carry flag is set if an ASCII number is found (Ø-9). This routine skips tabs, linefeeds, and spaces.

```
LD    HL,BUFPOS-1    ;Start position
                     ; in the buffer
RST   16             ;Parse buffer
JP    Z,ZERCLN       ;JP if 00H or colon
JP    C,ZTO9         ;JP if Ø thru 9
                     ; (30H-39H)
```

The Z80 cannot directly compare two 16-bit registers. The ROM has a built-in routine to perform such a function. The 'HL' register is compared to 'DE', and 'A' is lost. The flags are set depending on whether 'HL' is equal to, less than, or greater than 'DE'. If 'HL' = 'DE', the Z flag is set. If 'HL' < 'DE', the Carry flag is set. The interface is as follows:

```
        LD    HL,A          ;Load 1st number
        LD    DE,B          ;Load 2nd number
        RST   24            ;CP HL,DE
        JP    Z,AEB         ;JP if HL=DE
        JP    NC,AGTB       ;JP if HL>DE
        JP    C,ALTB        ;JP if HL<DE
```

If you need to delay for a period of time before continuing a process, simply load 'BC' with a count of 1AA7H for each tenth of a second and call DELAY:

```
              LD    BC,DLYCNT     ;Load delay count
        DELAY CALL  0060H         ;Delay
```

Ever want to route output to a device depending on a flag? The ROM routine DSPCHR does just that. It uses the OUTBFL at 409CH. If the flag has bit 7 set, output goes to the cassette. If not a zero (00H), output to the lineprinter. If equal to zero, output to the video. This routine maintains the CRTPOS at 40A6H (current line position on the video). However, this routine does have a catch (or in this case, hook). One of the first commands in this routine is a CALL to 41C1H. If you are running under vanilla Level II, this address should contain a RET. If you are under an OS, it may contain a JP instruction. BE CAREFUL!

```
              ;******
              ;* Check hook at 41C1H!
              ;* before calling DSPCHR
              ;******
              LD    A,CHAR        ;Load character to
                                  ; output
        DSPCHR CALL 032AH         ;Display char on
                                  ; current device
```

If you use the above routine, you may wish to use the RSTDEV routine at 038BH to restore the current device to the video display.

The INCHRS routine (at 0361H) is present which inputs up to 240 characters into a buffer using the BUFFIN routine at 05D9H. It places a 00H as the last character in the line to replace the carriage return. Upon exit, 'A' is zero if the input ended with an ENTER. The Carry is set if the BREAK was hit. 'HL' points to the beginning of the buffer minus one. Here we also find a hook to 41AFH. Be sure to consider this in your programming.

```
            ;******
            ;* Check hook at 41AF!
            ;******
            LD    HL,BUFFER     ;Address of input
                                ; buffer
INCHRS      CALL  0361H         ;Input max 240 chars
            INC   HL            ;HL to begin of
                                ; BUFFER
```

Another routine that uses INCHRS (361H) is QINPUT at 1BB3H. It will print a question mark and a space on the screen and then will accept up to 240 characters. The address of the pointer to the input buffer is in INBUFFP located at 40A7H. Remember the disk hook at 41AFH!

The following routine outputs the buffer pointed to by 'HL' to the current device using the DSPCHR routine, located at 032AH. The routine keeps outputting until it finds a 00H which marks the end of the data and which is not output.

```
            ;******
            ;* Check DSPCHR hook at 41C1H
            ;******
            LD    HL,BUFFER     ;Point to buffer
MSGOUT      CALL  2B75H         ;Output buffer until
                                ; a 00H
```

The next chapter contains items that didn't conveniently fit anywhere else. They include a discussion

of the RANDOM routine and several examples of real head scratching uses of Assembly language contained in the Level II ROM.

Random Ramblings

Did you ever wonder how the command RANDOM works? Well, first let's do a little test. Turn on your computer and go to Level II BASIC. Now, do a PRINT RND(0) and record the result (was it .768709?). Turn off the computer. Wait about 30 seconds and turn it back on. Go back into Level II do the same thing. You should get the same value. You see, there is no **random** number generator in the TRS-80 BASIC. It always starts at the same value upon power-up and proceeds to produce the numbers in the same series. Of course, the mathematical formula used to produce these "random" numbers gives us the impression that they are random since they are not in an obvious series.

The RND function uses a set algorithm to find the next random number, using a "seed" as the starting point. Well, BASIC has supplied a method of beginning at an undeterminable starting point by using the RANDOM command. However, after a bit of logical thinking, we come to the same conclusion: "How can it pick a RANDOM starting point if it has to do it using a set algorithm?" We could checksum memory (adding the contents of each byte and throwing away the carries), but even after that relatively long process, the end value may not be random (although it probably would be, in most cases). The solution is rather interesting.

Have you ever heard of the refresh register? You know, that funny register which we can't really use but is included in the instruction set. Oh yes, you remember; that's the one which has the constantly changing value that the CPU uses to refresh the memory chips! Starting to see the light?

RANDOM simply loads the accumulator with the value in the refresh register 'R' and stuffs it into part of the seed of the RND function. Look at the simple routine at 01D3H. This command is used so seldomly that some disassemblers don't even take it into consideration.

Remember looking at the disassembly of the ROM and seeing something like this:

```
0133 AF          XOR   A
0134 013E80      LD    BC,803EH
0137 013E01      LD    BC,013EH
```

It doesn't make a great deal of sense. Look carefully. Notice that the second byte of each of the LD BC instructions has a 3EH. Hmmmm. Coincidence? No, it is actually the Z80 opcode corresponding to a LD A,n. Let's take a look at how the source code could have looked:

```
0133 AF     ENT1  XOR   A            ;Flag=Z
0134 01           DEFB  01           ;Hide next instr with
                                     ; a LD BC
0135 3E80   ENT2  LD    A,80H        ;Flag=M
0137 01           DEFB  01           ;Hide next instr
0138 3E01   END3  LD    A,01H        ;Flag=NZ
```

This structure is used several times throughout the ROM to provide entry points to a single routine which set a different flag depending on which entry is used. The 01H could be changed to a 11H to hide the next instruction with a LD DE or a 21H for a LD HL. The code to use depends on which register pair does not contain information which should not be destroyed. Since the ROM is using HL and DE at most times, BC is the choice.

Another strange little bit of code one sometimes sees is like this:

```
PUSH  HL
POP   HL
PUSH  HL
POP   HL
```

This is not designed to make sure that the value is REALLY there. It is used as a short delay, usually for an I/O latch to be set before testing for status, such as with the floppy disk controller. Sometimes this routine will be replaced with

```
EX    (SP),HL                  EX    (SP),IX
EX    (SP),HL       or         EX    (SP),IX
```

```
EX    (SP),HL                    EX    (SP),IX
EX    (SP),HL                    EX    (SP),IX
```

which also function as delays, but use up a little more time. If you use these techniques, be SURE that each PUSH is matched with a corresponding POP and that the EXchanges are done in pairs. If you don't you will change the stack, creating unpredictable results.

Did you ever wonder why the HALT instruction (Op-code 76H) causes a reboot? HALT is normally used in an interrupt-driven machine to cause the CPU to halt execution until an interrupt occurs. The folks at Radio Shack have tied this signal to the Non Maskable Interrupt, so a HALT is the same as hitting RESET.

Did you ever want to disable the BREAK key in a BASIC program? It is rather simple to do. You see, BREAK is considered a "character" by the keyboard driver (the value of a BREAK is 01H). At the end of the driver at address 0453H, it checks to see if the character being returned is a BREAK (01H). If it is, it performs an RST 40, which under DOS is an entry to DEBUG. So, we simply change the RST vector to modify the value in the accumulator to whatever we want. Here's an example:

```
1000 BREAK=16396          '&H400C
1010 POKE BREAK,62        '&H3E (LD A,N)
1020 POKE BREAK+1,0       'Totally disable break
1030 POKE BREAK+2,201     'RET
1040 PRINT "BREAK HAS BEEN DISABLED."
```

How about this one:

```
1000 BREAK=16396          '&H400C
1010 POKE BREAK,62        '&H3E (LD A,N)
1020 POKE BREAK+1,191     'FLAG BREAK PRESSED
1030 POKE BREAK+2,201     'RET
1040 CLS: PRINT "Updating Database....":
          PRINT "      PLEASE do not hit BREAK"
1050 REM *** Update database here, checking keyboard
1060 REM *** input as follows:
1070 A$=INKEY$:
     IF A$=CHR$(191) then 2000
```

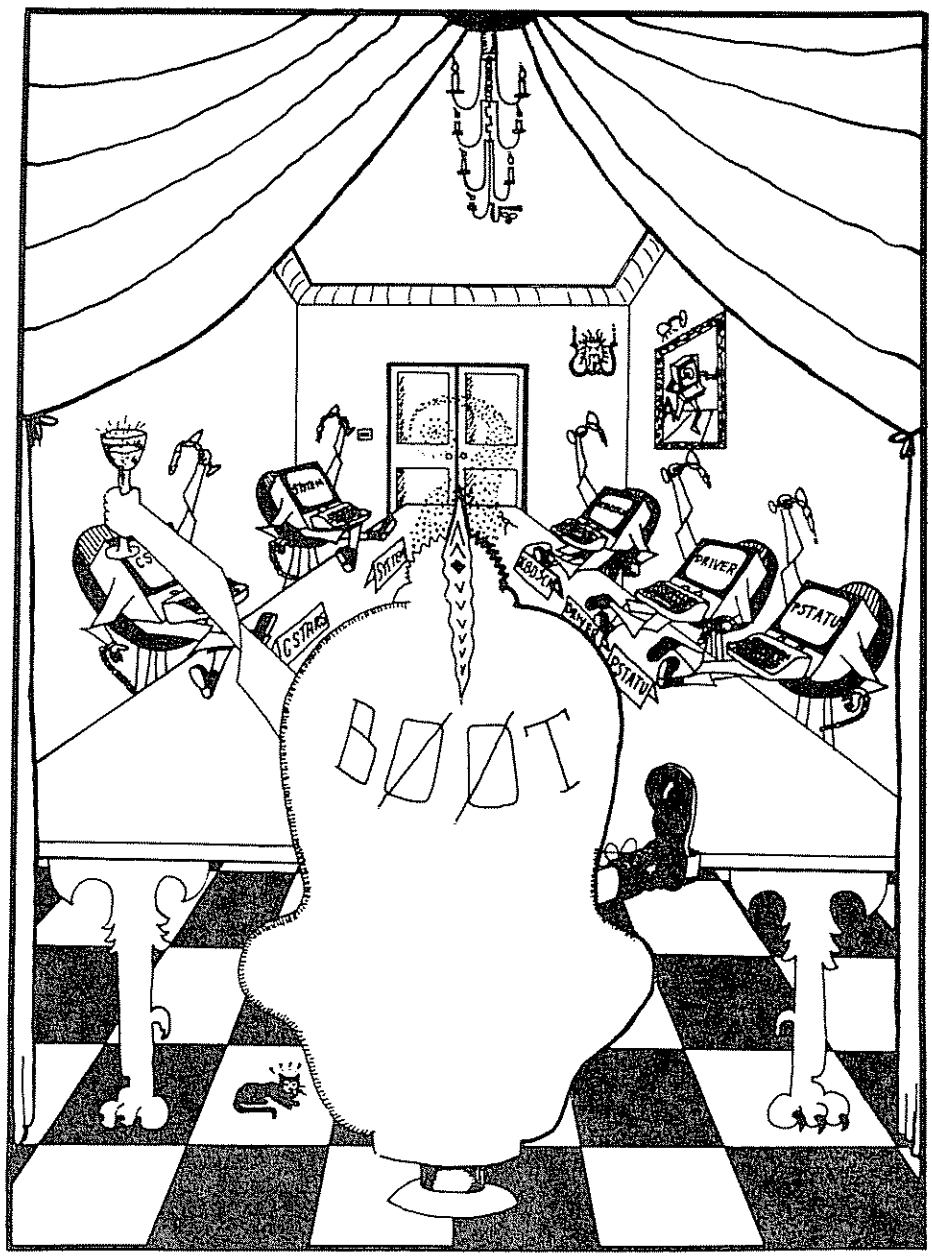
```
1080 REM *** Continue updating database
1090 POKE BREAK+1,1      'They've been good, restore BREAK
1100 PRINT "Thank you for you patience.":END
2000 CLS: PRINT "You obviously do not follow directions,":
          PRINT " even when you are asked politely!"
2010 PRINT "Just for that...":
     PRINT "   CLEARING DATABASE"
2020 REM *** Clear database here
2030 GOTO 2030
```

Somehow, I get the feeling that this may come back to haunt me...

Well, we seem to have run out of chapters at this point, but we still have enough Appendices to outfit a baseball team. There are several tables that we felt that you would need, and we have provided drivers for the I/O devices. They may not be fancy, but they do demonstrate the techniques that we have been discussing.

Appendix A: Label Table

The following list was developed to supply the assembly language programmer with a quick reference to routine entry points, I/O areas, storage areas, and pointers. It was <u>not</u> designed as a complete interfacing guide. Labels listed for the various addresses provide a meaningful code-word giving some indication of the use(s) of the routines or areas. Address locations not described in this volume are either self-explanatory, may be found in Radio Shack reference manuals, or are discussed in other volumes. This list is sorted alphabetically by labels. For a similar list, sorted by address, one can refer to Appendix A in Volume I.



START	END	LABEL	DESCRIPTION
1364	136B		DBL: 1D+10
136C	1373		DBL: 1D+15
1374	137B		DBL: 1D+16
1384	138B		DBL: 1D+16
07FD	0800		SNG: .598979
0801	0804		SNG: .981471
0805	0808		SNG: 2.88539
0834	0839		SNG: -.5 into BCDE
0841	0846		SNG: .693147 into BCDE
143C	1441		SNG: 1.4427
147A	147D		SNG: -1.41316E-4
147E	1481		SNG: 1.32988E-3
1482	1485		SNG: -8.30136E-3
1486	1489		SNG: .0416574
148A	148D		SNG: -0.166665
1594	1597		SNG: 39.7107
1598	159B		SNG: -76.575
159C	159F		SNG: 81.6022
15A0	15A3		SNG: -41.3417
15E4	15E7		SNG: 2.86623E-03
15E8	15EB		SNG: -0.0161657
15EC	15EF		SNG: 0.0429096
15F0	15F3		SNG: -0.0752896
15F4	15F7		SNG: 0.106563
15F8	15FB		SNG: -0.142089
15FC	15FF		SNG: 0.199936
1600	1603		SNG: -0.333331
37F0	37FF		Same as 37E0-37EF
403E	403F		Unused under Level II
404C	404F		Unused under Level II (interrupt processing under DOS)
407F			Unused under Level II
409F			Unused under Level II
40AD			Unused under Level II
4030	4032	ABORT	ABORT under DOS (unused under LII)
0977		ABS	ABS (Bcmd D9H)
0C5B	0C6F	ABSINT	Take absolute value of integer
4410		ACTINT	Activate an interrupt task
249F		ADD	+ (Bcmd CDH)
0C77		ADDDBL	Add double
070B		ADDHL	(HL) + FPA1 -> FPA1
0BD2		ADDINT	Integer add
0716		ADDSNG	Add single precision
298F		ADDSTR	Concatenate two strings
0708		ADHALF	FPA1 + 0.5 -> FPA1
25FD		AND	And (Bcmd D2H)
40FB	40FC	ARRAYS	Pointer to beginning of arrays
2A0F		ASC	Asc (Bcmd F6H)
0E65		ASCBIN	Convert ASCII buffer to binary value
4130		ASCBUF	Numeric work area: converted binary to ASCII number
0E6C		ASCINT	Convert ASCII buffer to integer value
0FBE		ASCUSG	Convert ASCII from 'USING' routine

START	END	LABEL	DESCRIPTION
15BD		ATN	Atn (Bcmd E4H)
15E3	1607	ATNTBL	Arctan data table
1A5A		ATOOFF	Turn AUTO off
40E4	40E5	AUTINC	Auto increment
2008		AUTO	Auto (Bcmd B7H)
40E1		AUTOFL	Auto flag (Non-zero=ON. Zero after BREAK)
40E2	40E3	AUTOLN	Auto line number
1A60		AUTOON	INC to new AUTO line number
06CC		BASIC	Proper re-entry to Level II BASIC
1650	1820	BCTBL	BASIC command table (b7 of 1st char. of reserved word high)
0FBD		BINASC	Convert binary value to ASCII
4445		BKSPA	Backspace a file
273D		BSERR	Subscript out of range error
05D9	0673	BUFFIN	Buffer input routine
0040		BUFFNV	Vector to buffer input routine (BUFFIN)
0000		CBOOT	ROM Level II Bootstrap
0ADB		CDBL	CDBL (Bcmd F1H)
4309		CDRVBT	Current drive being used with correct bit pattern already calculated and stored at this address.
0982		CHGSGN	Change sign routine
1963		CHKMEM	Check if enough memory available
0AF4		CHKSTR	Check type for string and TMERR if not
2A1F		CHR	Chr$ (Bcmd F7H)
0A7F		CINT	CINT (Bcmd EFH)
1E3D		CKA2Z	Check if a character A-Z
0955	0963	CKRMZP	Tests values for Minus, Zero, or Plus
1E7A		CLEAR	Clear (Bcmd B8H)
2C1F		CLOAD	CLoad (Bcmd B9H)
4185		CLOSE	CLOSE: (DBcmd A6H)
4428		CLOSE	CLOSE (DOS file call. P#6-11)
021E	022B	CLRCFF	Clear CFF
01C9		CLS	CLS (Bcmd 84H)
4173		CMD	CMD: (DBcmd 85H)
4405		CMDINT	Command Interpreter entry point
1822		CMDTBL	Entry points for command table (BCTBL)
1DE9		CNERR	Can't continue error
2169	2177	COFFIO	If cassette is on, turn off
0674	06CF	COLDST	Cold Start
37DF		COMDAT	Communication Data Address
4052	4053	COMINT	Communications interrupt vector
37DE		COMSTA	Communication Status Address
42E8		CON0	Constant: 0
1DE4		CONT	Cont (Bcmd B3H)
1E5A		CONVRT	Convert bytes in buffer to two-byte DE value
1541		COS	Cos (Bcmd E1H)
0A78		CPRDBL	Double precision compare
0A39		CPRINT	Integer compare
0A0C		CPRSNG	Compare single precision
0241	0260	CRBIT	Read bit from cassette
0235	0240	CRBYTE	Read byte from cassette
0296		CRLDR	Find sync., put stars in corner

START	END	LABEL	DESCRIPTION
401E	401F	CRTADR	Driver address (0458H)
0033		CRTBYT	Display byte in 'A' at cursor (DE lost)
4023	4024	CRTCON	Constant: D 0
401D	4024	CRTDCB	Video DCB
3C00	3FFF	CRTMEM	Video display memory
033A		CRTOUT	Output 'A' to video (DE saved)
40A6		CRTPOS	Current line position on Video
3C00	3C3F	CRTR1	Row 1 on CRT
3E40	3E7F	CRTR10	Row 10
3E80	3EBF	CRTR11	Row 11
3EC0	3EFF	CRTR12	Row 12
3F00	3F3F	CRTR13	Row 13
3F40	3F7F	CRTR14	Row 14
3F80	3FBF	CRTR15	Row 15
3FC0	3FFF	CRTR16	Row 16
3C40	3C7F	CRTR2	Row 2
3C80	3CBF	CRTR3	Row 3
3CC0	3CFF	CRTR4	Row 4
3D00	3D3F	CRTR5	Row 5
3D40	3D7F	CRTR6	Row 6
3D80	3DBF	CRTR7	Row 7
3DC0	3DFF	CRTR8	Row 8
3E00	3E3F	CRTR9	Row 9
401D		CRTTYP	DCB Type (07)
2BF5		CSAVE	CSave (Bcmd BAH)
37E4		CSELCT	Cassette select latch address
0AB1		CSNG	CSNG (Bcmd F0H)
022C	0234	CSTAR	Change star in corner for cassette operations
029F		CSTARS	Put stars in corner
403D		CSTATU	Cassette status byte
0075		CSTLII	Cold start for Level II BASIC
0023		CTLBYT	Output a control byte to a device.
01F8		CTOFF	Cassette off
01FE		CTON	Cassette on
0293	02A8	CTONRL	Cassette on, find sync., put stars in corner
0284	0292	CTONWL	Cassette on, write leader and sync. byte
430C	430D	CURBUF	Currently active I/O buffer for file reads/writes.
4022		CURCHR	Cursor character
430A	430B	CURDCB	Address of currently active DCB
4308		CURDRV	Current drive being used
40A2	40A3	CURLIN	Current line number
40EC	40ED	CURNUM	Current line number
430E		CUROVL	Current overlay in memory
4020	4021	CURPOS	Cursor position on screen (L,H)
40D8	40D9	CURTKN	Stores pointer to current token
415E		CVD	CVD: (DBcmd E8H)
4152		CVI	CVI: (DBcmd E6H)
4158		CVS	CVS: (DBcmd E7H)
0261	0283	CW2BYT	Write byte to cassette twice
01D9	01F7	CWBIT	Write bit to cassette
0264		CWBYT	Write byte to cassette

START	END	LABEL	DESCRIPTION
0287		CWLDR	Write leader and sync. byte
199A		DOERR	Division by zero error
1F05		DATA	Data (Bcmd 88H)
40A9		DATAFL	Data statement flag
4470		DATE	Returns DATE into 8-byte HL buffer
40FF	4100	DATPTR	Pointer to delimeter after last DATA Value read
37EF		DATREG	Floppy disk data register
4045		DAY	Day
2CA5	2CA8	DBAD	Data "BAD<CR>"
4152	41A5	DBJPVS	Disk BASIC jump vectors
0D33	0D44	DBLMA	Double precision mantissa addition
0D45	0D56	DBLMS	Double precision mantissa subtract
0AB9		DBLSNG	Convert double to single
000D		DBOOT	Vector to disk bootstrap
1930	1934	DBREAK	Data "Break"
4419		DCTTSK	Deactivate an interrupt task
2733		DDERR	Redimensioned array error
440D		DEBUG	Enter the real-time debugging facility
405D		DEBUG1	Debug: A or H (ASCII or H) or LSB of first breakpoint
405E		DEBUG2	Debug: 0=Normal screen, <>0 = Full screen: or MSB of first BREAKPT
405F		DEBUG3	Debug: Instruction byte at breakpoint
4060	4061	DEBUG4	Debug: Second breakpoint or single-step
4062		DEBUG5	Debug: Instruction byte at second breakpoint
4063	4064	DEBUG6	Debug: Address currently being displayed on screen
4065	407C	DEBUGS	DEBUG: Register save area (AF,BC,DE,HL,AF´,BC´,DE´,HL´,IX,IY,SP,PC)
4315	4317	DEBUGV	Debug vector
415B		DEF	DEF: (DBcmd B0H)
1E09		DEFDBL	Defdbl (Bcmd 9BH)
0212	021D	DEFDRV	Define cassette drive from ´A´
4473		DEFEXT	Add default file extension
1E03		DEFINT	Defint (Bcmd 99H)
1E06		DEFSNG	Defsng (Bcmd 9AH)
1E00		DEFSTR	Defstr (Bcmd 98H)
0060	0065	DELAY	Delay routine (BC=Counter. 14.66 msec/loop)
2BC6		DELETE	Delete (Bcmd B6H)
2286	2294	DEXTIG	Data "?Extra ignored"
2608		DIM	Dim (Bcmd 8AH)
4300	4307	DIRTRK	Locations of the directory tracks of the different drives
069F		DISKBT	Disk bootstrap
0897		DIV10	FPA1 / 10 -> FPA1
0DE5		DIVDBL	Double precision division
2490		DIVINT	Integer divide
4080	408D	DIVRAM	RAM used with single precision divide
08A2		DIVSNG	Divide single precision
0F18		DIVTEN	Divide by ten (10)
40AE		DLFLG	Dimension/Let flag from parser
0105	0110	DMEMSZ	Data "MEMORY SIZE"
4318	4347	DOSBUF	DOS Command buffer
4200	42FF	DOSIOB	DOS I/O buffer for sectors from disk

START	END	LABEL	DESCRIPTION
4049	404A	DOSMEM	DOS memory size determined at power-up
402D*	402F	DOSVEC	DOS Transfer Vector
1928	192E	DREADY	Data "READY<CR>"
2178	217D	DREDO	Data "?REDO"
03C2	03E2	DRIVER	I/O Driver
0046		DRIVRV	Vector to I/O driver routine @ 03C2H
0111	012B	DRSL2B	Data "RADIO SHACK LEVEL II BASIC<CR>"
37E1		DSELCT	Disk drive select latch address
407D	407E	DSKBSP	Disk boot stack pointer beginning location
5200	6FFF	DSKUTL	Disk BASIC/DOS utilities/User memory
032A	0347	DSPCHR	Display byte on current device (Device flg @ 409CH)
2E60		EDIT	Edit (Bcmd 9DH)
1F07		ELSE	Else (Bcmd 95H)
1DAE		END	End (Bcmd 80H)
51FF		ENDOVR	End of DOS overlay area
40FD	40FE	ENDVAR	End location of array variables
1A19		ENTLII	Entry point to Level II BASIC
4161		EOF	EOF: (DBcmd E9H)
41A6		ERHOOK	Hook to Disk BASIC for long error msgs.
24DD		ERL	Erl (Bcmd C2H)
24CF		ERR	Err (Bcmd C3H)
40F2		ERRFLG	FFH after error. Zero if no error
40EA	40EB	ERRLIN	Line containing error
409A		ERRNBR	Level II Error
1FF4		ERROR	Error (Bcmd 9EH)
40F0	40F1	ERRPRC	Address of "ON ERROR"
19A2		ERRPRT	Output an error msg
18C9	18F6	ERRTBL	Error abbreviation table
1439		EXP	Exp (Bcmd E0H)
1479	1499	EXPTBL	Exp data table
4125		EXPWRK	Exponent work area
1E4A		FCERR	Illegal function call error
37EC		FDCADR	Floppy disk controller address
218A		FDERR	Bad file data error
417C		FIELD	FIELD: (DBcmd A3H)
0B26		FIX	Fix (Bcmd F2H)
4155		FN	FN: (DBcmd BEH)
249F		FNSCAN	Scan for functions
1CA1		FOR	For (Bcmd 81H)
40DC		FORFLG	Set to 64 on FOR loop. Prevent subscripted variable.
4121	4124	FPA1	Floating Point Accumulator
4124		FPA1E	Characteristic (exponent)
0778		FPA1EZ	Zero exponent of FPA1
4121	4123	FPA1M	Mantissa
4127	412E	FPA2	Floating Point Accumulator #2
09CB		FPAMEM	Transfer FPA1 to (HL)
27D4		FRE	Fre (Bcmd DAH)
1608		FUNTBL	Function Table
417F		GET	GET: (DBcmd A4H)
0314	031C	GETADR	Get a 2 byte address from tape (Ret in HL)
0049	004F	GETCHR	Scan keyboard waiting for input. (DE lost)

START	END	LABEL	DESCRIPTION
1E4F	1E79	GETLN	Scan line for line number
1EB1		GOSUB	Gosub (Bcmd 91H)
1EC2		GOTO	Goto (Bcmd 8DH)
0132	01C8	GRPHCS	Graphics Routines
02A9		GSYSTR	Get transfer address for system
0384	038A	GTDCHR	Get one char. input from keyboard. (DE saved)
1E4F	1E79	GTLNUM	Get line number
441C		GTSPEC	Get a file specification from buffer
148E	1491	HALF	SNG: .5
158B	158E	HALFPI	SNG: 1.5708 (PI/2)
137C	1384	HLFDBL	DBL: .5
09B1		HLFPA1	(HL) --> FPA1
0ACF		HLSNG	Convert HL to single
189A		HRCHY	Algebraic heirarchy table
4043		HRS	Hours
2831		IDERR	Illegal direct error
2039		IF	If (Bcmd 8FH)
40A7	40A8	INBUFP	Input buffer pointer
0013		INBYT	Input a byte from a device
0361	0383	INCHRS	Input up to 240 chars. into 'HL' buffer. End of line has zero byte.
1B4D		INIT	Initialize work area
4420		INIT	INIT (DOS file call. P#6-8)
019D		INKEY	Inkey$ (Bcmd C9H)
2AEF		INP	Inp (Bcmd DBH)
4093	4095	INPRAM	INP function (93 = "IN" instruction, 94 = port, 95 = Ret)
219A		INPUT	Input (Bcmd 89H)
419D		INSTR	INSTR: (DBcmd C5H)
0B37		INT	Int (Bcmd D8H)
0B59	0B9D	INTDBL	Take integer of double
404C		INTENB	Interrupts enabled (bit mask)
37E0		INTLAT	Interrupt Latch Address
404B		INTMSK	Interrupt mask
0B3D	0B58	INTSNG	Take integer of single
404D	405C	INTTBL	Interrupt jump address for interrupts 0-7
41E6	42E7	IOBUFF	I/O Buffer
4033	4035	IODERR	Called by driver after illogical driver call
3801		KB1	Location for: @ A B C D E F G
3802		KB2	Location for: H I J K L M N O
3804		KB3	Location for: P Q R S T U V W
3808		KB4	Location for: X Y Z
3810		KB5	Location for: 0 1 2 3 4 5 6 7
3820		KB6	Location for: 8 9 : ; , - . / (Also ()*+<=>?)
3840		KB7	Location for: Enter Clear Break Arrow D.Arrow L.Arrow R.Arrow Space
4018	401C	KBCONS	Constant: 0 0 0 K I
4016	4017	KBDADR	Driver address (03E3H)
0358	0360	KBDSCN	Scan keyboard. (DE NOT LOST)
4036		KBIM1	01H
4037		KBIM2	02H
4038		KBIM3	04H

START	END	LABEL	DESCRIPTION
4039		KBIM4	08H
403A		KBIM5	10H
403B		KBIM6	20H
403C		KBIM7	40H
4036	403C	KBIMAG	Keyboard image
002B		KBSCAN	Keyboard scan return input in A. (DE lost.)
0050	005F	KBTBL	Table of Special Characters for keyboard routine
4015		KBTYP	DCB Type (01)
4099		KEYBUF	Inkey$ buffer or flag (last key hit on keyboard)
4015	401C	KEYDCB	Keyboard DCB
03E3	0457	KEYIN	Keyboard scan driver
3800	3BFF	KEYMEM	Keyboard memory (1,2,4,8,10,20,40,80H)
4191		KILL	KILL: (DBcmd AAH)
442C		KILL	KILL (DOS file call. P#6-11)
0000	2FFF	L2ROM	Radio Shack Level II BASIC ROM
4000	4014	L2VECS	Level II fixed RAM vectors
012D		L3ERR	Level III error
7FFF		LAD16K	Last RAM address in a 16K TRS-80
BFFF		LAD32K	Last RAM address in a 32K TRS-80
FFFF		LAD48K	Last RAM address in a 48K TRS-80
4FFF		LAD4K	Last RAM address in a 4K TRS-80
09BF		LDFPA1	Load FPA1 into BCDE
09C2		LDFPHL	Load real value pointed to by HL
2A61		LEFT	Left$ (Bcmd F8H)
2A03		LEN	Len (Bcmd F3H)
1F21		LET	Let (Bcmd 8CH)
41A3		LINE	LINE: (DBcmd 9CH)
409D		LINLEN	Maximum length of a line on the screen
2B2E		LIST	List (Bcmd B4H)
40E6	40E7	LLEND	Points to end of previous line or current line
2B29		LLIST	LList (Bcmd B5H)
4188		LOAD	LOAD: (DBcmd A7H)
4430		LOAD	Load a machine language format file
4164		LOC	LOC: (DBcmd EAH)
4167		LOF	LOF: (DBcmd EBH)
0809		LOG	Log (Bcmd DFH)
4047	4048	LOW	Contains address of lowest byte of avail. mem under DOS
039C	03C1	LPDCHR	Output byte in 'A' to printer (DE saved)
2067		LPRINT	Lprint (Bcmd AFH)
37E8		LPTADR	Line printer address
4026	4027	LPTADR	Driver address (058DH)
003B		LPTBYT	Send byte in 'A' to printer (DE lost)
402A	402C	LPTCON	Constant: 0 P R
4025	402C	LPTDCB	Lineprinter DCB
058D	05D8	LPTDRV	Printer driver
4029		LPTLCT	Line counter
4028		LPTLPP	Number of lines/page
409B		LPTPOS	Line printer line position
4025		LPTTYP	DCB Type (06)
29A3		LSERR	String too long error
4197		LSET	LSET: (DBcmd ABH)

START	END	LABEL	DESCRIPTION
40B1	40B2	LSTBYT	Address of last usable byte in memory (BASIC)
40DA	40DB	LSTDTL	Last data line number read
27C9		MEM	Mem (Bcmd C8H)
00C4	00D5	MEMSIZ	Determine memory size
418B		MERGE	MERGE: (DBcmd A8H)
2A9A		MID	Mid$ (Bcmd FAH)
4042		MINS	Minutes
0AA3	0AA8	MINVAL	SNG: -32768 / BCDE
4170		MKD	MKD$: (DBcmd EEH)
416A		MKI	MKI$: (DBcmd ECH)
416D		MKS	MKS$: (DBcmd EDH)
4046		MO	Month
24A0		MOERR	Missing operand error
09D3		MOVDAT	Move data from (HL) --> (DE)
2B75		MSGOUT	Output a msg until zero (0)
0DA1		MULDBL	Double precision multiply
0BF2		MULINT	Integer multiply
0847		MULSNG	Multiply single precision
418E		NAME	NAME: (DBcmd A9H)
40F7	40F8	NBIBP	Ptr to next byte to be used with "CONT"
1492	1495	NEGONE	SNG: -1.0
1B49		NEW	New (Bcmd BBH)
22BC		NEXT	Next (Bcmd 87H)
199D		NFFRR	Next without For error
0066	0074	NMI	Non-maskable interrupt
1A76		NOAUTO	Auto-off line input
2FC4		NOT	Not (Bcmd CBH)
198A		NRERR	No resume error
2212		ODERR	Out of data error
22A0		ODERR2	Out of data error (also @ 2212H)
197A		OMERR	Out of memory error
1F6C		ON	On (Bcmd A1H)
07F8	07FB	ONE1	SNG: 1.0
1496	1499	ONE2	SNG: 1.0
1604	1607	ONE3	SNG: 1.0
4179		OPEN	OPEN: (DBcmd A2H)
4424		OPEN	OPEN (DOS file call. P#6-9)
4476		OPTION	Get optional command flags from buffer
25F7		OR	Or (Bcmd D3H)
28DB		OSERR	Out of string space error
2AFB		OUT	Out (Bcmd A0H)
409C		OUTBFL	Output bit flag: 0=Video, 1=Lp, 80=Cassette
001B		OUTBYT	Output a byte to a device
20FE		OUTCR	Output a CR to current device
4467		OUTLIN	Output a line to the CRT
28A7		OUTLN	Output a line until zero (0)
446A		OUTLP	Output a line to the printer
4096	4098	OUTRAM	OUT function (96=Out,97=port,98=Ret)
07B2		OVERR	Overflow error
430F		OVLDBG	Overlay/Debug flag
13D8	13E1	P10TAB	Power of ten table: 10000,1000,100,10,1

START	END	LABEL	DESCRIPTION
2C8A	2C92	PBAD	Prints "BAD" on screen
2CAA		PEEK	Peek (Bcmd E5H)
227C	2285	PEXTIG	Load "?Extra ignored"
40A4	40A5	PGMBGN	Pointer to start of BASIC program
40EE	40EF	PLEND	Pointer to previous line end
0132		POINT	Point (Bcmd C6H)
2CB1		POKE	Poke (Bcmd B1H)
08A0		POPFPA	Restores old BCDE from stack
27F5		POS	Pos (Bcmd DCH)
4448		POSEOF	Position a file to EOF
0348	0357	POSIND	Line position indicator
4442		POSN	POSN (DOS file call. P#6-9)
4409		POSTER	Post error message entry point
13F2		POWER	Raise to a power (Ex: X raised to the N, X**N)
206F		PRINT	Print (Bcmd B2H)
409E		PRNTZN	Next print zone (reached after a comma as in ?A,B,C)
05D1		PSTATU	Test printer status. Z Flag set if ready.
40D8		PUCBYT	Printusing control byte: Bit2=*,3=+,4=$,6=Comma
4182		PUT	PUT: (DBcmd A5H)
1BB3		QINPUT	Print "? ". Input up to 240 characters
158F	1592	QUARTR	SNG: .25
01D3		RANDOM	Random (Bcmd 86H)
40DE		RDINFL	Read/Input flag: Non-zero=read / Zero = input
21EF		READ	Read (Bcmd 8BH)
4436		READ	READ (DOS file call. P#6-9)
1A25		READY	Load "READY" message
411D	4124	REAL8	Double precision variable
411D	4120	REAL8M	Extended mantissa : Double precision
		REM	Rem (Bcmd 93H)
0138		RESET	Reset (Bcmd 82H)
40FF	40A0	RESTLN	Used with RESTORE. Keeps current line number for "READ"
1D91		RESTOR	Restore (Bcmd 90H)
1FAF		RESUME	Resume (Bcmd 9FH)
1EDE		RETURN	Return (Bcmd 92H)
443F		REWIND	Rewind a file to the beginning
1EEA		RGERR	Return without Gosub error
2A91		RIGHT	Right$ (Bcmd F9H)
14C9		RND	Rnd (Bcmd DEH)
4090	4092	RNDMUL	Mantissa of multiplicative constant for RND
40AA	40AC	RNSEED	RND function seed
419A		RSET	RSET: (DBcmd ACH)
0010		RST16	INC HL. If (HL) is ASCII 0-9 SCF.
			If value is zero, set Z flag. Skips spaces.
1D78	1D90	RST16	Inc HL/ If (HL) is ASCII 0-9 SCF.
4003		RST16	RST16: 1D78; INC HL/If ASCII 0-9 SCF/Set if Z/Skip Spa
0018		RST24	CP HL,DE (A lost.)
1C90	1C95	RST24	CP HL,DE (A lost)
4006		RST24	RST24: 1C90; CP HL,DE (A lost.)
0020		RST32	P/U TYPFLG at 40AFH. If <8 SCF. RRT.
			Flags set as a result of type. M=Int.,Z=Str,PO=SNG,NC=DBL
25D9		RST32	From RST 32: P/U flag @ 40AFH. If <8 SCF, RRT.

START	END	LABEL	DESCRIPTION
4009		RST32	RST32: 25D9; If TYPFLG<8, SCF/RRT/M=INT,Z=STR,PO=SNG,NC=DBL
0028		RST40	JP DOS command processor
400C		RST40	RST40: DOS Command Processor
0030		RST48	Debug breakpoint
400F		RST48	RST48: Debug breakpoint
0038		RST56	Interrupt Mode 1
4012		RST56	RST56: Interrupt mode 1
0008		RST8	(Parser) CP (Syntax)/RST16 if =/Else SNERR
1C96	1CA0	RST8	(Parser) CP (Syntax). RST16 if equal. Else SNERR.
4000		RST8	RST8: 1C96; (Parser) CP (Syntax)/RST16 IF=/Else SNERR
038B	039B	RSTDEV	Reset devices. Set output back to CRT
06D2	06DD	RSTRTS	RST's loaded into RAM starting @ 4000H
4040		RTSC	25 MSec Real-time scheduling counter
1EA3		RUN	Run (Bcmd 8EH)
4433		RUN	Load and execute machine language file
19A0		RWERR	Resume without error
41A0		SAVE	SAVE: (DBcmd ADH)
0A9A		SAVINT	Save integer in HL to FPA1. Vartyp -> Int (2)
158B	15A7	SCDTBL	Sin/Cos data table
40F9	40FA	SCLERS	Pointer to beginning of scalers
37EE		SECREG	Floppy disk sector register
4041		SECS	Seconds
0135		SET	Set (Bcmd 83H)
0AEC		SETDBL	Change type flag to DBL
0A9D		SETINT	Change TYPFLG to INT
0AEF		SETSNG	Change type flag to single
098A		SGN	SGN (Bcmd D7H)
098D		SGNAE	Alternate entry point to SGN
3880		SHIFT	Location for: Shift (Electric pencil control key @ 10H)
1547		SIN	Sin (Bcmd E2H)
1593	15A7	SINTBL	Sin data table
1997		SNERR	Syntax error
09B4		SNGFPA	BCDE (Single precision val.) --> FPA1
0ACC		SNGINT	Convert integer to single
1BC0		SPACK	Source pack routine
40E8	40E9	SPSAV	Stack pointer save area
13E7		SQR	SQR (Bcmd DDH)
0814	0819	SQR2O2	SNG: .707107 (SQR(2)/2) into BCDE
0221		STATFF	Change status of CFF from HL
2B01		STEP	Step (Bcmd CCH)
28A1		STERR	String formula too complex error
2B7E		STFUNP	Scan text until zero. Unpack into INBUFP buffer
09A4	09B0	STKFP1	Puts a real value onto the stack
1DA9		STOP	Stop (Bcmd 94H)
40B5	40D2	STPRMS	String param. area. 3 byte sets. 1ST=Length, 2-3=Address
2836		STR	Str$ (Bcmd F4H)
40D4	40D5	STRADR	Address of current string
40D6	40D7	STRFRE	Next free byte in string area
2A2F		STRING	String$ (Bcmd C4H)
40D3		STRLEN	Length of current string
40A0	40A1	STRNGS	Beginning of string area

START	END	LABEL	DESCRIPTION
40B3	40B4	STRPTR	String parameter pointer
2532		SUB	- (Bcmd CEH)
0C70		SUBDBL	Subtract double
0710		SUBHL	(HL) - FPA1 -> FPA1
0BC7		SUBINT	Integer subtract
0713		SUBSNG	Subtract single precision
031D	0329	SYSGO	Jump to system start address
02B2		SYSTEM	System entry point
2137		TAB	Tab((Bcmd BCH)
15A8		TAN	Tan (Bcmd E3H)
0DD4	0DDB	TENDBL	DBL: 10.0
4176		TIME	TIME$: (DBcmd C7H)
446D		TIME	Move current TIME to 8-byte HL buffer
0AF6		TMERR	Type mismatch error
40DF	40E0	TRAADR	Transfer address for system
411B		TRCFLG	TRON - AF, TROFF - 0
37ED		TRKREG	Floppy disk track register
1DF8		TROFF	Troff (Bcmd 97H)
1DF7		TRON	Tron (Bcmd 96H)
4300	5FFF	TRSDOS	DOS routines
4416		TSKCHG	Change state of an interrupt task
4413		TSKOFF	Turn off an interrupt task
154A	154F	TWOPI	SNG: 6.28319 (2 PI) / BCDE
15A4	15A7	TWOPI	SNG: 6.28319 (2 PI)
40B0		TYPFL2	Variable type for FPA2
40AF		TYPFLG	Current variable type (8=DBL, 4=SGL, 3=STR, 2=INT)
4101	411A	TYPTBL	Variable types for each letter A-Z
2003		UEERR	Unprintable error
1ED9		ULERR	Undefined line error
2CBD		USING	Using (Bcmd BFH)
27FE		USR	Usr (Bcmd C1H)
408E	408F	USRADR	USR function address
0A7F		USRINP	Put 'USR' function argument in HL
0A9A		USROUT	Make HL output of 'USR' call
2AC5		VAL	Val (Bcmd F5H)
24ED		VARPTR	Varptr (Bcmd C0H)
443C		VERIFY	Write and verify a file write
0458	058C	VIDEO	Video display driver
000B		WHERE	Resolve Relocation Address

Appendix B: Lowercase Driver

```
01000 ;***********************************************
01010 ;*      LC:     Lower Case Driver              *
01020 ;*              Copyright (c) 1980             *
01030 ;*              Insiders Software Consultants  *
01040 ;*              PO Box 2441, Dept. LC          *
01050 ;*              Springfield, VA   22152        *
01060 ;***********************************************
01070 ;********
01080 ;*      This lowercase driver is intended to replace the
01090 ;*      driver currently offered by Radio Shack.  It
01100 ;*      provides UPPER/lower case, auto repeat, debounce,
01110 ;*      JKL (screen print option), print switch (output to
01120 ;*      screen and line printer at the same time), re-boot
01130 ;*      switch, and control key (@) providing control codes
01135 ;*      from the keyboard.
01140 ;*      Current definition of special characters:
01150 ;*              Sh. CLEAR=Underline
01160 ;*              CTL-CLEAR=Lineprinter switch
01170 ;*              Sh. BREAK=UPPER CASE LOCK
01180 ;*              CTL-BREAK=REBOOT!
01190 ;*              CTL-UPARR=Circumflex (^)
01200 ;*              CTL-L.ARR=Left Curly ({)
01210 ;*              Up Arr    =Left bracket
01220 ;*              Sh. R.ARR=Right Bracket (])
01230 ;*              CTL-R.ARR=Right Curly (})
01240 ;*              CTL-Zero =At-sign (@)
01250 ;*              CTL-1     =FS (1CH)
01260 ;*              CTL-2     =GS (1DH)
01270 ;*              CTL-3     =RS (1EH)
01280 ;*              CTL-4     =US (1FH)
01290 ;*              CTL-5     =Backslash
01300 ;*              CTL-6     =OR (7CH)
01310 ;*              CTL-7     =DEL (7FH)
01320 ;*              CTL-8     =Tilde (^)
01330 ;*              CTL-9     =Pause (Used to be shift-@)
01340 ;* Note:  The codes can be changed to suit your needs
01350 ;*       by changing the values in the ASCII table.
01360 ;********
```

```
401E            01370 CRTADR  EQU     401EH           ;Video Driver Address
4016            01380 KBDADR  EQU     4016H           ;Keyboard Driver Addr.
7000            01390         ORG     7000H           ;MAY BE CHANGED
7000            01400 PLC     EQU     $               ;Entry point
                01410
                01420 ;********
                01430 ;*      The next two lines MUST BE ADDED if you are
                01440 ;*      running under the NEWDOS (not NEWDOS80) disk
                01450 ;*      operating system.  They disable the NEWDOS
                01460 ;*      JKL function to avoid a conflict between the
                01470 ;*      DOS and the lowercase driver
                01480 ;********
                01490 ;       LD      HL,43B5H
                01500 ;       LD      (HL),0C9H               ;NEWDOS ONLY!
                01510
                01520 ;********
                01530 ;*      Check for bit-6 static RAM chip present.
                01540 ;*      If it is not, no lowercase display.
                01550 ;*      Machine will be locked into UPPERCASE only.
                01560 ;********
7000 21003C     01570         LD      HL,3C00H        ;Begin of video memory
7003 46         01580         LD      B,(HL)          ;Get value at location
7004 3EFF       01590         LD      A,0FFH          ;All bits set in 'A'
7006 77         01600         LD      (HL),A          ;Store in video mem
7007 BE         01610         CP      (HL)            ;CP mem w/ value stored
7008 70         01620         LD      (HL),B          ;Restore original value
7009 2821       01630         JR      Z,LCMOD         ;If =, RAM #6 present.
                01640 ;********
                01650 ;*      No LC mod present or active.  Disable lowercase
                01660 ;*      functions
                01670 ;********
700B 215F71     01680         LD      HL,UCLS+1       ;Address of conversion JR
700E 7E         01690         LD      A,(HL)          ;Get value
700F EE06       01700         XOR     6               ;Set uppercase mode
7011 77         01710         LD      (HL),A          ;LOCK UPPER
7012 3EC9       01720         LD      A,0C9H          ;LD A, RET
7014 32DD71     01730         LD      (UCLOCK),A      ;No LOCK toggle
7017 3EC3       01740         LD      A,0C3H          ;LD A,JP
7019 32D470     01750         LD      (NOLP),A        ;Disable new VIDEO driver
701C 3E58       01760         LD      A,58H           ;Place a JP 0458H to the
701E 32D570     01770         LD      (NOLP+1),A      ;Old video driver
7021 3E04       01780         LD      A,4
7023 32D670     01790         LD      (NOLP+2),A
7026 1804       01800         JR      LCMOD           ;Execute relocation
```

```
7028 0000      01810 DIFF    DEFW    0                ;Relocation constant
702A 0000      01820 LOAD    DEFW    0                ;Load address
702C 019501    01830 LCMOD   LD      BC,ZEND-START    ;Length of driver
702F 2AB140    01840         LD      HL,(40B1H)       ;Get top of memory
               01850                                  ;For DOS, change to 4049H
7032 B7        01860         OR      A
7033 ED42      01870         SBC     HL,BC            ;Determine load address
7035 222A70    01880         LD      (LOAD),HL        ;Store load address
7038 01BA70    01890         LD      BC,START         ;Load the start address
703B B7        01900         OR      A
703C ED42      01910         SBC     HL,BC            ;Determine relocation
                                                      ; constant
703E 222870    01920         LD      (DIFF),HL        ;Store relocation constant
               01930 ;********
               01940 ;*      This next section uses a relocation table to
               01950 ;*      adjust the absolute addresses found in the
               01960 ;*      lowercase driver.  The table contains the address
               01970 ;*      of a hex address that must be adjusted before the
               01980 ;*      program is moved to its new location in high
               01990 ;*      memory.  This is done by adding the relocation
               02000 ;*      constant to the current value, and restoring the
               02010 ;*      new value.
               02020 ;********
7041 DD216970  02030 RELOC   LD      IX,RTABLE        ;Load addr of table
7045 DD6601    02040 RELOCT  LD      H,(IX+1)         ;P/U MSB of address
7048 DD6E00    02050         LD      L,(IX)           ;P/U LSB of address
704B 7D        02060         LD      A,L              ;Check for end-table=0000H
704C B4        02070         OR      H
704D 2838      02080         JR      Z,DRELOC         ;If end, DONE
704F E5        02090         PUSH    HL               ;LD IY,HL
7050 FDE1      02100         POP     IY
7052 FD6E00    02110         LD      L,(IY)           ;Get absolute address
               02120                                  ; from memory
7055 FD6601    02130         LD      H,(IY+1)
7058 ED5B2870  02140         LD      DE,(DIFF)        ;Get relocation constant
705C 19        02150         ADD     HL,DE            ;Correct abs. addr.
705D FD7500    02160         LD      (IY),L           ;Address back to memory
7060 FD7401    02170         LD      (IY+1),H
7063 DD23      02180         INC     IX               ;Next table location
7065 DD23      02190         INC     IX
7067 18DC      02200         JR      RELOCT           ;Loop till end of table
```

```
                02210 ;********
                02220 ;*      Relocation Table
                02230 ;********
7069 0671       02240 RTABLE  DEFW    REL1+1
706B 0D71       02250         DEFW    REL2+2
706D 1171       02260         DEFW    REL3+2
706F 3C71       02270         DEFW    REL4+2
7071 4071       02280         DEFW    REL5+2
7073 8D71       02290         DEFW    REL6+1
7075 A271       02300         DEFW    NOSHFT+1
7077 B471       02310         DEFW    REL7+2
7079 B871       02320         DEFW    REL8+2
707B C371       02330         DEFW    REL9+1
707D DE71       02340         DEFW    UCLOCK+1
707F E771       02350         DEFW    LPT00+1
7081 AC70       02360         DEFW    REL0+1
7083 B270       02370         DEFW    REL10+1
7085 0000       02380         DEFW    0               ;End of table
                02390
                02400 ;********
                02410 ;*      Relocation Complete
                02420 ;********
7087 2A2A70     02430 DRELOC  LD      HL,(LOAD)       ;Get load address
708A 2B         02440         DEC     HL              ;Get new top of memory
708B 22B140     02450         LD      (40B1H),HL      ;Save TOPMEM
708E 224940     02460         LD      (4049H),HL      ;Save HIGH$
7091 11CEFF     02470         LD      DE,-50
7094 19         02480         ADD     HL,DE           ;CLEAR 50
7095 22A040     02490         LD      (40A0H),HL      ;Save string area
                02500 ;********
                02510 ;*      Move the program to the new location
                02520 ;********
7098 ED5B2A70   02530         LD      DE,(LOAD)       ;Get destination
709C 21BA70     02540         LD      HL,START        ;LD start address
709F 019501     02550         LD      BC,ZEND-START   ;LD length of driver
70A2 EDB0       02560         LDIR                    ;MOVE!
70A4 3ADD71     02570         LD      A,(UCLOCK)      ;Is there LC mod?
70A7 FEC9       02580         CP      0C9H
70A9 2806       02590         JR      Z,REL10         ;JP if no LC
                02600 ;********
                02610 ;*      Place the address of the new video driver
                02620 ;*      in the video DCB to activate driver.
                02630 ;********
70AB 21BE70     02640 REL0    LD      HL,VPATCH       ;New video driver address
70AE 221E40     02650         LD      (CRTADR),HL     ;Place in video DCB
```

```
                    02660 ;********
                    02670 ;*      Patch the keyboard DCB with the address
                    02680 ;*      of the new keyboard driver
                    02690 ;********
70B1 21F270         02700 REL10   LD      HL,KPATCH       ;Address of keyboard driver
70B4 221640         02710         LD      (KBDADR),HL     ;SAVE NEW KBRD DRIVER
70B7 C3CC06         02720         JP      06CCH           ;Done. GOTO BASIC

                    02730 ;********
                    02740 ;*      Lowercase driver
                    02750 ;********
70BA                02760 START   EQU     $               ;Start address
70BA 0000           02770 SAVBC   DEFW    0               ;'BC' save area
70BC 0138           02780 SAVDE   DEFW    3801H           ;'DE' save area
4018                02790 CNTR    EQU     4018H           ;Counter address
0500                02800 PER1    EQU     500H            ;Repeat counter
                    02810                                 ; # loops before 1st repeat
0050                02820 PER2    EQU     50H             ;Inter-character repeat
                                                          ; count
                    02830 ;********
                    02840 ;*      Repeat counters may be adjusted up or down to
                    02850 ;*      suit the taste of the user.  If repeat is too
                    02860 ;*      fast, increase value.  If repeat is too slow,
                    02870 ;*      decrease value.
                    02880 ;********
3880                02890 SHIFT   EQU     3880H           ;Shift key location
3801                02900 CONTRL  EQU     3801H           ;Control key= @
70BE 1814           02910 VPATCH  JR      NOLP            ;Printer switch @ Vpatch+1
70C0 C5             02920         PUSH    BC              ;Output to printer too
70C1 F5             02930         PUSH    AF
70C2 79             02940         LD      A,C             ;Get char to output
70C3 FE20           02950         CP      20H             ;Check for control codes
70C5 3008           02960         JR      NC,LPIT         ;If not carry, output char
70C7 FE07           02970         CP      7
70C9 3807           02980         JR      C,NOVALD        ;If <7, not valid
70CB FE0E           02990         CP      0EH
70CD 3003           03000         JR      NC,NOVALD       ;If >13, not valid
70CF CD3B00         03010 LPIT    CALL    3BH             ;Output to LP
70D2 F1             03020 NOVALD  POP     AF
70D3 C1             03030         POP     BC
70D4 DD6E03         03040 NOLP    LD      L,(IX+3)        ;Video patch for LC letters
70D7 DD6604         03050         LD      H,(IX+4)        ;Get cursor location
70DA DA9A04         03060         JP      C,49AH          ;Jp if control code
70DD DD7E05         03070         LD      A,(IX+5)        ;Get cursor character
70E0 B7             03080         OR      A
70E1 2801           03090         JR      Z,$+3           ;Skip next if no char
70E3 77             03100         LD      (HL),A          ;Restore char at cursor
70E4 79             03110         LD      A,C             ;Restore character
70E5 FE20           03120         CP      20H             ;JP if control code
                    03130                                 ; to ROM driver
```

```
70E7 DA0605    03140          JP      C,506H
70EA FE80      03150          CP      80H
70EC D2A604    03160          JP      NC,4A6H          ;Jump to 04A6 if >80
70EF C37D04    03170          JP      47DH             ;Entry to printing chars

               03180 ;********
               03190 ;*       Keyboard driver
               03200 ;********
70F2 213640    03210 KPATCH   LD      HL,4036H         ;KB Save area
70F5 110138    03220          LD      DE,3801H         ;KB Scan area
70F8 0E00      03230          LD      C,0              ;Init key counter
70FA 1A        03240 KLOOP    LD      A,(DE)           ;Scan keyboard search for
               03250                                   ; new characters
70FB 47        03260          LD      B,A              ;Save scan value
70FC AE        03270          XOR     (HL)             ;Check for previous
70FD 70        03280          LD      (HL),B           ;Store new scan
70FE A0        03290          AND     B
70FF 2032      03300          JR      NZ,KPRSD         ;JP IF NEW KEY
7101 0C        03310          INC     C                ;Next count
7102 2C        03320          INC     L                ;Next location decode
                                                        ; matrix
7103 CB03      03330          RLC     E                ;Next addr in KEYMEM
7105 F2FA70    03340 REL1     JP      P,KLOOP          ;Loop if not 8 done
7108 2A1840    03350          LD      HL,(CNTR)        ;Test repeat
710B ED4BBA70  03360 REL2     LD      BC,(SAVBC)       ;Get old BC
710F ED5BBC70  03370 REL3     LD      DE,(SAVDE)       ;Get old DE
7113 1A        03380          LD      A,(DE)           ;Get key scan
7114 A0        03390          AND     B
7115 2007      03400          JR      NZ,CHKCNT        ;See if still pressed
7117 210005    03410          LD      HL,PER1          ;No key. RESET count
711A 221840    03420          LD      (CNTR),HL        ;Reset counter
711D C9        03430          RET
711E 2B        03440 CHKCNT   DEC     HL               ;Check counter
711F 08        03450          EX      AF,AF'
7120 7C        03460          LD      A,H
7121 B5        03470          OR      L
7122 2806      03480          JR      Z,RPT            ;If =0, repeat key
7124 08        03490          EX      AF,AF'
7125 221840    03500          LD      (CNTR),HL        ;Save counter
7128 AF        03510          XOR     A                ;Value ret=0
7129 C9        03520          RET
712A 08        03530 RPT      EX      AF,AF'
712B 215000    03540          LD      HL,PER2
712E 221840    03550          LD      (CNTR),HL        ;Set PER2 count
7131 1806      03560          JR      KPRSD1           ;Key repeat
7133 210005    03570 KPRSD    LD      HL,PER1          ;Restore repeat to PER1
7136 221840    03580          LD      (CNTR),HL
7139 47        03590 KPRSD1   LD      B,A
713A ED43BA70  03600 REL4     LD      (SAVBC),BC       ;Save 'BC' for repeat
713E ED53BC70  03610 REL5     LD      (SAVDE),DE       ;Save 'DE' for repeat
```

```
                03620 ;********
                03630 ;*      Decode matrix value
                03640 ;********
7142 CB01       03650         RLC     C               ;*2
7144 CB01       03660         RLC     C               ;*4
7146 CB01       03670         RLC     C               ;*8
7148 0C         03680 KINC    INC     C
7149 0F         03690         RRCA
714A 30FC       03700         JR      NC,KINC
714C 0D         03710         DEC     C               ;Adjust for over-add
714D 3A8038     03720         LD      A,(SHIFT)       ;See if <SHIFT>
7150 47         03730         LD      B,A             ;Store for later
7151 3A0138     03740         LD      A,(CONTRL)      ;See if control
7154 5F         03750         LD      E,A             ;Store for later
7155 79         03760         LD      A,C             ;Restore and adjust value
7156 C640       03770         ADD     A,40H
7158 FE60       03780         CP      60H
715A 3011       03790         JR      NC,NOALPH       ;JP if non-alpha
715C CB0B       03800         RRC     E
715E 3004       03810 UCLS    JR      NC,NOCTRL       ;Uppercase lock here
7160 D640       03820         SUB     40H             ;Convert uppercase
7162 1845       03830         JR      FINIS           ;Done
7164 4F         03840 NOCTRL  LD      C,A
7165 CB08       03850         RRC     B               ;Test for shift
7167 3840       03860         JR      C,FINIS         ;JP if shift
7169 EE20       03870         XOR     20H             ;Case switch
716B 183C       03880         JR      FINIS
                03890 ;********
                03900 ;*      Adjust control codes and test for LOOKUP
                03910 ;********
716D D670       03920 NOALPH  SUB     70H             ;Non-alpha
716F 3020       03930         JR      NC,LOOKUP
7171 C640       03940         ADD     A,40H
7173 FE3C       03950         CP      3CH
7175 3802       03960         JR      C,NOCHG
7177 EE10       03970         XOR     10H
7179 CB08       03980 NOCHG   RRC     B
717B 3004       03990         JR      NC,CKCTRL
717D EE10       04000         XOR     10H
717F 1828       04010         JR      FINIS
7181 CB0B       04020 CKCTRL  RRC     E               ;Check for control (@)
7183 3024       04030         JR      NC,FINIS        ;Done if no control
7185 FE3A       04040         CP      3AH
7187 3020       04050         JR      NC,FINIS
7189 D62F       04060         SUB     2FH
718B C8         04070         RET     Z
718C 210672     04080 REL6    LD      HL,TABLE2-1
718F 1813       04090         JR      NOSHFT+3
```

```
                04100 ;********
                04110 ;*      Lookup key scan in table.
                04120 ;********
7191 57         04130 LOOKUP  LD      D,A             ;TABLE LOOKUP ROUTINE
7192 07         04140         RLCA
7193 82         04150         ADD     A,D
7194 CB08       04160         RRC     B
7196 3003       04170         JR      NC,CTRL1
7198 3C         04180         INC     A               ;Shift character
7199 1806       04190         JR      NOSHFT
719B CB0B       04200 CTRL1   RRC     E
719D 3002       04210         JR      NC,NOSHFT
719F 3C         04220         INC     A               ;Control character
71A0 3C         04230         INC     A
71A1 21EF71     04240 NOSHFT  LD      HL,TABLE        ;Begin of table
71A4 5F         04250         LD      E,A
71A5 1600       04260         LD      D,0
71A7 19         04270         ADD     HL,DE           ;Add offset in 'A'
71A8 7E         04280         LD      A,(HL)          ;Get value
71A9 F5         04290 FINIS   PUSH    AF              ;ALMOST FINISHED
71AA 013303     04300         LD      BC,0333H        ;Debounce count
71AD 0B         04310 DBNCE   DEC     BC
71AE 78         04320         LD      A,B
71AF B1         04330         OR      C
71B0 20FB       04340         JR      NZ,DBNCE
71B2 ED4BBA70   04350 REL7    LD      BC,(SAVBC)
71B6 ED5BBC70   04360 REL8    LD      DE,(SAVDE)
71BA 1A         04370         LD      A,(DE)          ;Key still pressed?
71BB A0         04380         AND     B
71BC 2003       04390         JR      NZ,NOBNCE       ;JP if YES
71BE F1         04400         POP     AF              ;Key bounced
71BF AF         04410         XOR     A               ;No character
71C0 C9         04420         RET
71C1 F1         04430 NOBNCE  POP     AF              ;Good key
71C2 CD1172     04440 REL9    CALL    JKL             ;SCREEN PRINT?
71C5 4F         04450         LD      C,A
71C6 B7         04460         OR      A
71C7 F0         04470         RET     P               ;RETURN IF NO SPECIAL CHAR.
                04480 ;********
                04490 ;*      Special characters have bit 7 set.  The special
                04500 ;*      command number is determined by what OTHER bit
                04510 ;*      is also set by rotating right until a carry.
                04520 ;********
71C8 0F         04530         RRCA
71C9 3812       04540         JR      C,UCLOCK        ;81H Is uppercase lock
71CB 0F         04550         RRCA
71CC 3818       04560         JR      C,LPTOO         ;82H is lineprinter switch
71CE 0F         04570         RRCA
71CF 300A       04580         JR      NC,NEXT         ;JR if not 84H, REBOOT
```

```
71D1 3A4038    04590 WAITBR  LD      A,(3840H)       ;Wait for <BREAK> released
71D4 E604      04600         AND     4
71D6 20F9      04610         JR      NZ,WAITBR       ;Loop if <BREAK> down
71D8 C30000    04620         JP      0000H           ;84H    BOOT.
               04630 ;********
               04640 ;*      Note: Could define keys 88H,90H,0A0H,0C0H as
               04650 ;*      other special keys by performing a RRCA and
               04660 ;*      jumping on CARRY to the processing routine
               04670 ;********
71DB AF        04680 NEXT    XOR     A               ;Key undefined
71DC C9        04690         RET
               04700 ;********
               04710 ;*      Uppercase lock/unlock toggle
               04720 ;********
71DD 215F71    04730 UCLOCK  LD      HL,UCLS+1       ;JR offset location
71E0 7E        04740         LD      A,(HL)
71E1 EE06      04750         XOR     6
71E3 77        04760         LD      (HL),A
71E4 AF        04770         XOR     A               ;Return no value
71E5 C9        04780         RET
               04790 ;********
               04800 ;*      Line printer toggle
               04810 ;********
71E6 21BF70    04820 LPTOO   LD      HL,VPATCH+1     ;Video patch JR offset
71E9 7E        04830         LD      A,(HL)
71EA EE14      04840         XOR     14H
71EC 77        04850         LD      (HL),A          ;XOR with offset
71ED AF        04860         XOR     A               ;Return no value
71EE C9        04870         RET
               04880 ;********
               04890 ;*      Keyboard Lookup Table
               04900 ;*              Format=
               04910 ;*                      Key unshifted
               04920 ;*                      Key Shifted
               04930 ;*                      Key w/ CONTROL
               04940 ;********
71EF 0D        04950 TABLE   DEFB    0DH     ;ENTER
71F0 0D        04960         DEFB    0DH
71F1 0D        04970         DEFB    0DH
71F2 1F        04980         DEFB    1FH     ;CLEAR
71F3 5F        04990         DEFB    5FH     ;       (UNDERLINE)
71F4 82        05000         DEFB    82H     ;       (LPTOO)
71F5 01        05010         DEFB    01H     ;BREAK
71F6 81        05020         DEFB    81H     ;       (UCLOCK)
71F7 84        05030         DEFB    84H     ;BOOT!
71F8 5B        05040         DEFB    5BH     ;UP ARR.(L BRACK)
71F9 1B        05050         DEFB    27      ;ESC
71FA 5E        05060         DEFB    5EH     ;       (Circum)
```

```
71FB 0A       05070          DEFB    0AH     ;D. ARR.(CNTRL)
71FC 1A       05080          DEFB    1AH     ;       (LF)
71FD 00       05090          DEFB    0
71FE 08       05100          DEFB    8H      ;L. ARR (BSP)
71FF 18       05110          DEFB    18H     ;       (CANCEL)
7200 7B       05120          DEFB    7BH     ;       (L CURLY)
7201 09       05130          DEFB    9       ;R. ARR (TAB)
7202 5D       05140          DEFB    5DH     ;       (R BRACK)
7203 7D       05150          DEFB    7DH     ;       (R CURLY)
7204 20       05160          DEFB    20H     ;SPACE
7205 20       05170          DEFB    20H
7206 20       05180          DEFB    20H
              05190 ;********
              05200 ;*       Special table for control 0-9
              05210 ;********
7207 40       05220 TABLE2   DEFB    40H     ;CTL0-AT
7208 1C       05230          DEFB    1CH     ;CTL1-FS
7209 1D       05240          DEFB    1DH     ;   2-GS
720A 1E       05250          DEFB    1EH     ;   3-RS
720B 1F       05260          DEFB    1FH     ;   4-US
720C 5C       05270          DEFB    5CH     ;   5-BACK SLASH
720D 7C       05280          DEFB    7CH     ;   6-OR
720E 7F       05290          DEFB    7FH     ;   7-DEL
720F 7E       05300          DEFB    7EH     ;   8-TILDE
7210 60       05310          DEFB    60H     ;   9-PAUSE
              05320 ;********
              05330 ;*       JKL:  Presing J,K,& L keys at the same time will
              05340 ;*                send what is on the screen to the
              05350 ;*                printer.
              05351 ;*       Break, Clear, and Enter will abort output
              05360 ;********
7211 E5       05380 JKL      PUSH    HL              ;Save 'HL'
7212 67       05390          LD      H,A
7213 3A0238   05400          LD      A,(3802H)
7216 FE1C     05410          CP      1CH             ;JKL?
7218 7C       05420          LD      A,H
7219 E1       05430          POP     HL
721A C0       05440          RET     NZ              ;Return if no JKL
721B E5       05450          PUSH    HL
721C D5       05460          PUSH    DE
721D 21003C   05470          LD      HL,3C00H        ;First screen location
7220 7D       05480 JKLOOP   LD      A,L
7221 E63F     05490          AND     3FH
```

```
7223 3E0D      05500 JKLCR   LD      A,0DH
7225 CC3B00    05510         CALL    Z,3BH           ;If EOL, output CR
7228 7E        05520         LD      A,(HL)          ;Get screen value
7229 23        05540         INC     HL
722A FE80      05560         CP      80H
722C 3802      05570         JR      C,JKLOUT
722E 3E2E      05580         LD      A,'.'           ;Replace graphics with '.'
7230 F5        05590 JKLOUT  PUSH    AF
7231 3A4038    05600         LD      A,(3840H)
7234 E607      05610         AND     7
7236 2006      05620         JR      NZ,JKLSTP       ;Stop if ENT/CLR/BRK
7238 F1        05630         POP     AF
7239 CD3B00    05640         CALL    3BH             ;Output character
723C 18E2      05650         JR      JKLOOP          ;Loop til done
723E 3A4038    05660 JKLSTP  LD      A,(3840H)
7241 E607      05670         AND     7
7243 20F9      05680         JR      NZ,JKLSTP       ;Wait til abort released
7245 3E0D      05690         LD      A,0DH           ;Send CR
7247 CD3B00    05700         CALL    3BH
724A F1        05710         POP     AF
724B D1        05720 JKLST   POP     DE
724C E1        05730         POP     HL
724D AF        05740         XOR     A
724E C9        05750         RET                     ;JKL done
724F           05760 ZEND    EQU     $               ;End of program
7000           05770         END     PLC
00000 Total errors
```

CHKCNT	711E	CKCTRL	7181
CONTRL	3801	CRTADR	401E
DBNCE	71AD	DIFF	7028
FINIS	71A9	JKL	7211
JKLOOP	7220	JKLOUT	7230
JKLSTP	723E	KBDADR	4016
KLOOP	70FA	KPATCH	70F2
KPRSD1	7139	LCMOD	702C
LOOKUP	7191	LPIT	70CF
NEXT	71DB	NOALPH	716D
NOCHG	7179	NOCTRL	7164
NOSHFT	71A1	NOVALD	70D2
PER2	0050	PLC	7000
REL1	7105	REL10	70B1
REL3	710F	REL4	713A
REL6	718C	REL7	71B2
REL9	71C2	RELOC	7041
RPT	712A	RTABLE	7069
SAVDE	70BC	SHIFT	3880
TABLE	71EF	TABLE2	7207
UCLS	715E	VPATCH	70BE
ZEND	724F		

CNTR	4018
CTRL1	719B
DRELOC	7087
JKLCR	7223
JKLST	724B
KINC	7148
KPRSD	7133
LOAD	702A
LPTOO	71E6
NOBNCE	71C1
NOLP	70D4
PER1	0500
REL0	70AB
REL2	710B
REL5	713E
REL8	71B6
RELOCT	7045
SAVBC	70BA
START	70BA
UCLOCK	71DD
WAITBR	71D1

Appendix C: Alternate System

```
                    01000 ;****************************************
                    01010 ;*      Alternate SYSTEM Loader         *
                    01020 ;*                                      *
                    01030 ;*      By Insiders Software            *
                    01040 ;*              Consultants, Inc.       *
                    01050 ;*      PO Box 2441, Dept. SYS          *
                    01060 ;*      Springfield, Virginia 22152     *
                    01070 ;****************************************
41E2                01080           ORG     41E2H           ;SYSTEM Hook
41E2 C31453         01090           JP      START           ;Patch to execute
                    01100                                   ; new loader.
                    01110 ;********
                    01120 ;*      ROM routines that we will use
                    01130 ;********
021E                01140 CLRCFF    EQU     021EH           ;CLR Cassette Flip Flop
01F8                01150 CTOFF     EQU     01F8H           ;Cassette off
01FE                01160 CTON      EQU     01FEH           ;Cassette on
032A                01170 DSPCHR    EQU     032AH           ;Display character
1BB3                01180 QINPUT    EQU     1BB3H           ;Print ? in input
06CC                01190 BASIC     EQU     06CCH           ;BASIC entry point
1997                01200 SNERR     EQU     1997H           ;Syntax Error
022C                01210 CSTAR     EQU     022CH           ;Change star in corner
1E5A                01220 CONVRT    EQU     1E5AH           ;DEC ASCII to BINARY
40DF                01230 TRXADR    EQU     40DFH           ;Transfer Address location
4288                01240 STACK     EQU     4288H           ;Stack pointer set
20FE                01250 OUTCR     EQU     20FEH           ;Output a CR to device
                    01260 ;********
                    01270 ;*      SYSTEM
                    01280 ;********
40DF                01290           ORG     TRAADR
40DF CC06           01300           DEFW    06CCH           ;Default transfer to BASIC
3C00                01310           ORG     3C00H
3C00 2A             01320           DEFM    '**************************************'
     2A 2A 2A 2A 2A 2A 2A
     2A 2A 2A 2A 2A 2A 2A 2A
     2A 2A 2A 2A 2A 2A 2A 2A
     2A 2A 2A 2A 2A 2A 2A 2A
     2A 2A 2A
```

```
3C40                01330           ORG     3C40H
3C40 2A             01340           DEFM    '*      Alternate SYSTEM Loader     *'
     20 20 20 20 20 20 41 6C
     74 65 72 6E 61 74 65 20
     53 59 53 54 45 4D 20 4C
     6F 61 64 65 72 20 20 20
     20 20 2A
3C80                01350           ORG     3C80H
3C80 2A             01360           DEFM    '*  Insiders Software Consultants   *'
     20 20 49 6E 73 69 64 65
     72 73 20 53 6F 66 74 77
     61 72 65 20 43 6F 6E 73
     75 6C 74 61 6E 74 73 20
     20 20 2A
3CC0                01370           ORG     3CC0H
3CC0 2A             01380           DEFM    '**********************************'
     2A 2A 2A 2A 2A 2A 2A 2A
     2A 2A 2A 2A 2A 2A 2A 2A
     2A 2A 2A 2A 2A 2A 2A 2A
     2A 2A 2A 2A 2A 2A 2A 2A
     2A 2A 2A
5200                01390           ORG     5200H
5200 CD7552         01400 SYSENT    CALL    GETADR          ;Get address from tape
5203 22DF40         01410           LD      (TRAADR),HL     ;Load into transfer addr
5206 CDF801         01420           CALL    CTOFF           ;Cassette off
5209 318842         01430 SYSTEM    LD      SP,STACK        ;Set stack pointer
520C CDFE20         01440           CALL    OUTCR           ;Output a CR
520F 219552         01450           LD      HL,SYSCMD       ;Command msg
5212 CD8C52         01460           CALL    OUTLIN          ;Output msg
5215 CDB31B         01470           CALL    QINPUT          ;Get input from user
5218 DACC06         01480           JP      C,BASIC         ;BASIC if <BREAK>
521B D7             01490           RST     16              ;Test buffer
521C CA9719         01500           JP      Z,SNERR         ;If nothing, syntax err
521F FE2F           01510           CP      '/'             ;Goto address
5221 CA7E52         01520           JP      Z,SYSGO
5224 E5             01521           PUSH    HL
5225 21A652         01522           LD      HL,TITLE
5228 CD8C52         01523           CALL    OUTLIN
522B E1             01524           POP     HL
522C CD2053         01550           CALL    CTONRL          ;Cassette on/read leader
522F CD3653         01560 GETTL     CALL    CRBYTE
5232 FE55           01570           CP      55H             ;Search for title byte
5234 20F9           01580           JR      NZ,GETTL        ;Loop for title
5236 0606           01590           LD      B,6             ;Max num chars in title
5238 CD3653         01640 GETTL1    CALL    CRBYTE          ;Get title byte
523B 2A2040         01641           LD      HL,(4020H)      ;CURSOR POSITION
523E 77             01642           LD      (HL),A
523F 23             01643           INC     HL
5240 222040         01644           LD      (4020H),HL
5243 10F3           01660           DJNZ    GETTL1
```

```
5245 CD2C02   01670 GETREC  CALL    CSTAR           ;Change star
5248 CD3653   01680 GETBLK  CALL    CRBYTE          ;Get byte
524B FE78     01690         CP      78H             ;Byte preceeding trans. adr
524D CA0052   01700         JP      Z,SYSENT
5250 FE3C     01710         CP      3CH             ;Byte preceeding load adr
5252 20F4     01720         JR      NZ,GETBLK       ;Loop til block marker
5254 CD3653   01730         CALL    CRBYTE          ;Get number of bytes
5257 47       01740         LD      B,A             ;Store number
5258 CD7552   01750         CALL    GETADR          ;Get load address
525B 85       01760         ADD     A,L             ;Add load addr to cksum
525C 4F       01770         LD      C,A             ;Save cksum in 'C'
525D CD3653   01780 BLKRD   CALL    CRBYTE          ;Get data byte
5260 77       01790         LD      (HL),A          ;Store in mem
5261 23       01800         INC     HL              ;Inc load address
5262 81       01810         ADD     A,C             ;Add previous cksum
5263 4F       01820         LD      C,A             ;Save new cksum
5264 10F7     01830         DJNZ    BLKRD           ;Loop thru data
5266 CD3653   01840         CALL    CRBYTE          ;Get cksum
5269 B9       01850         CP      C               ;Cksum match?
526A 28D9     01860         JR      Z,GETREC
526C 21BA52   01870         LD      HL,CKSUM
526F CD8C52   01880         CALL    OUTLIN          ;Cksum error
5272 C30952   01890         JP      SYSTEM
5275 CD3653   01900 GETADR  CALL    CRBYTE          ;Get address from tape
5278 6F       01910         LD      L,A             ;Save LSB
5279 CD3653   01920         CALL    CRBYTE          ;Get MSB
527C 67       01930         LD      H,A
527D C9       01940         RET
527E EB       01950 SYSGO   EX      DE,HL           ;System execute to addr
527F 2ADF40   01960         LD      HL,(TRAADR)     ;Get transfer address
5282 EB       01970         EX      DE,HL           ;Put in DE
5283 D7       02000         RST     16
5284 C45A1E   02010         CALL    NZ,CONVRT       ;Convert to val in DE
5287 C20952   02020         JP      NZ,SYSTEM
528A EB       02030         EX      DE,HL           ;Switch addr to HL
528B E9       02040         JP      (HL)            ;Goto routine!
528C 7E       02060 OUTLIN  LD      A,(HL)          ;Output a line to video
528D B7       02070         OR      A               ;End?
528E C8       02080         RET     Z
528F CD2A03   02090         CALL    DSPCHR
5292 23       02100         INC     HL
5293 18F7     02110         JR      OUTLIN
5295 0A0A     02120 SYSCMD  DEFW    0A0AH           ;Linefeeds
5297 53       02130         DEFM    'SYSTEM Command'
     59 53 54 45 4D 20 43 6F
     6D 6D 61 6E 64
52A5 00       02140         DEFB    0
```

```
52A6 0A0A      02150 TITLE   DEFW    0A0AH
52A8 50        02160         DEFM    'Program loading: '
     72 6F 67 72 61 6D 20 6C
     6F 61 64 69 6E 67 3A 20
52B9 00        02170         DEFB    0
52BA 0D        02180 CKSUM   DEFB    0DH
52BB 43        02190         DEFM    'Checksum error!'
     68 65 63 6B 73 75 6D 20
     65 72 72 6F 72 21
52CA 0D00      02200         DEFW    0DH
52CC 0A0A      02210 STRTM   DEFW    0A0AH
52CE 0A0A      02220         DEFW    0A0AH
52D0 0A0A      02230         DEFW    0A0AH
52D2 41        02240         DEFM    'Alternate SYSTEM Loader'
     6C 74 65 72 6E 61 74 65
     20 53 59 53 54 45 4D 20
     4C 6F 61 64 65 72
52E9 0D        02250         DEFB    0DH
52EA 20        02260         DEFM    '  by Insiders Software Consultants, Inc.'
     20 62 79 20 49 6E 73 69
     64 65 72 73 20 53 6F 66
     74 77 61 72 65 20 43 6F
     6E 73 75 6C 74 61 6E 74
     73 2C 20 49 6E 63 2E
5312 0D00      02270         DEFW    0DH
5314 CDC901    02280 START   CALL    01C9H           ;Clear screen
5317 21CC52    02290         LD      HL,STRTM
531A CD8C52    02300         CALL    OUTLIN          ;Welcome message
531D C30952    02310         JP      SYSTEM
5320 CDFE01    02320 CTONRL  CALL    CTON            ;Cassette on/read leader
5323 E5        02330         PUSH    HL
5324 AF        02340         XOR     A
5325 CD4253    02350 CRLDR   CALL    CRBIT           ;Read bit
5328 FEA5      02360         CP      0A5H            ;Sync. byte
532A 20F9      02370         JR      NZ,CRLDR
532C 3E2A      02380         LD      A,'*'           ;Stars in corner
532E 323E3C    02390         LD      (3C3EH),A       ;Put first star
5331 323F3C    02400         LD      (3C3FH),A       ;Put second star
5334 E1        02410         POP     HL              ;Restore HL
5335 C9        02420         RET
5336 C5        02430 CRBYTE  PUSH    BC              ;Read byte from cassette
5337 E5        02440         PUSH    HL
5338 0608      02450         LD      B,8
533A CD4253    02460 CRBYTL  CALL    CRBIT           ;Read bit from cassette
533D 10FB      02470         DJNZ    CRBYTL          ;8 bits=byte
533F E1        02480         POP     HL
5340 C1        02490         POP     BC
5341 C9        02500         RET
```

```
5342 C5       02510 CRBIT   PUSH    BC              ;Read a bit
5343 F5       02520         PUSH    AF
5344 DBFF     02530 SRTIMB  IN      A,(0FFH)        ;Search for timing bit
5346 17       02540         RLA
5347 30FB     02550         JR      NC,SRTIMB       ;If no bit, try again
5349 0641     02560         LD      B,41H           ;First timing loop
534B 10FE     02570         DJNZ    $               ;Delay
534D CD1E02   02580         CALL    CLRCFF          ;Clear cassette Flip-flop
5350 0650     02590         LD      B,50H           ;Second delay
5352 10FE     02600         DJNZ    $               ;Delay
5354 0614     02610         LD      B,14H           ;Redundant bit read
5356 DBFF     02620 RBITR   IN      A,(0FFH)        ;Get bit
5358 10FC     02630         DJNZ    RBITR           ;Loop
535A 47       02640         LD      B,A
535B F1       02650         POP     AF              ;Restore 'A'
535C CB10     02660         RL      B               ;Rotate high into carry
535E 17       02670         RLA                     ;Get bit to low order
535F F5       02680         PUSH    AF
5360 CD1E02   02690         CALL    CLRCFF          ;Clear CFF
5363 F1       02700         POP     AF
5364 C1       02710         POP     BC
5365 C9       02720         RET
5314          02730         END     START
```

SYSTEM Loader

00000 Total errors

BASIC	06CC	BLKRD	525D	CKSUM	52BA
CLRCFF	021E	CONVRT	1E5A	CRBIT	5342
CRBYTE	5336	CRBYTL	533A	CRLDR	5325
CSTAR	022C	CTOFF	01F8	CTON	01FE
CTONRL	5320	DSPCHR	032A	GETADR	5275
GETBLK	5248	GETREC	5245	GETTL	522F
GETTLI	5238	OUTCR	20FE	OUTLIN	528C
QINPUT	1BB3	RBITR	5356	SNERR	1997
SRTIMB	5344	STACK	4288	START	5314
STRTM	52CC	SYSCMD	5295	SYSENT	5200
SYSGO	527E	SYSTEM	5209	TITLE	52A6
TRAADR	40DF				

Appendix D: ASCII Table

Character	Hex	Decimal	ASCII Use	TRS-80 Use
* * * * *	Communications Control Characters			* * * * *
NUL	00H	00	NULL	NULL
SOH	01H	01	Start of Heading	Break
STX	02H	02	Start of Text	None
ETX	03H	03	End of Text	None
EOT	04H	04	End of Transmission	None
ENQ	05H	05	Enquiry	None
ACK	06H	06	Acknowledge	None
BEL	07H	07	Bell or Alarm	None
BS	08H	08	Backspace	Backspace & erase previous character
HT	09H	09	Horizontal Tab	Horizontal Tab
LF	0AH	10	Linefeed	Translated to Carriage-return
VT	0BH	11	Vertical Tab	Translated to Carriage-return
FF	0CH	12	Form feed (Top of Page)	Translated to Carriage-return
CR	0DH	13	Carriage-return	Carriage-return
SO	0EH	14	Shift Out of Standard Character Set	Turn on Cursor
SI	0FH	15	Shift Into Standard Character Set	Turn off Cursor
DLE	10H	16	Data Link Escape	None
DC1	11H	17	Device Control 1 (Transmit On)	None
DC2	12H	18	Device Control 2 (Paper Tape On)	None
DC3	13H	19	Device Control 3 (Transmit Off)	None
DC4	14H	20	Device Control 4 (Paper Tape Off)	None
NAK	15H	21	Negative Acknowledge	None
SYN	16H	22	Synchronous Idle	None
ETB	17H	23	End of Transmission	Convert to 32-Character Mode
CAN	18H	24	Cancel	Erase Input Buffer
EM	19H	25	End of Medium	Advance Cursor

Character	Hex	Decimal	ASCII Use	TRS-80 Use
SUB	1AH	26	Substitute	Linefeed
ESC	1BH	27	Escape (Alt Mode)	Upward Linefeed
FS	1CH	28	File Separator	Return Cursor to Upper Left Corner of Screen
GS	1DH	29	Group Separator	Move Cursor to Start of Line
RS	1EH	30	Record Separator	Erase to End of Line
US	1FH	31	Unit Separator	Erase to End of Screen

* * * * * Special Characters and Numbers * * * * *

Character	Hex	Decimal	Character Name
Space	20H	32	Space
!	21H	33	Exclamation Point
"	22H	34	Quotation Mark
#	23H	35	Number (Pound) Sign
$	24H	36	Dollar Sign
%	25H	37	Percent
&	26H	38	Ampersand
'	27H	39	Apostrophe
(	28H	40	Open Parenthesis
)	29H	41	Close Parenthesis
*	2AH	42	Asterisk
+	2BH	43	Plus Sign
,	2CH	44	Comma
-	2DH	45	Hyphen (Minus)
.	2EH	46	Period
/	2FH	47	Slash
0	30H	48	Zero
1	31H	49	One
2	32H	50	Two
3	33H	51	Three
4	34H	52	Four
5	35H	53	Five
6	36H	54	Six
7	37H	55	Seven
8	38H	56	Eight
9	39H	57	Nine
:	3AH	58	Colon
;	3BH	59	Semicolon
<	3CH	60	Less-than Sign
=	3DH	61	Equal Sign
>	3EH	62	Greater-than Sign
?	3FH	63	Question Mark

Character	Hex	Decimal	Character	Hex	Decimal
Upper-Case Alphabet			**Lower-Case Alphabet**		
@	40H	64	Accent Grave	60H	96
A	41H	65	a	61H	97
B	42H	66	b	62H	98
C	43H	67	c	63H	99
D	44H	68	d	64H	100
E	45H	69	e	65H	101
F	46H	70	f	66H	102
G	47H	71	g	67H	103
H	48H	72	h	68H	104
I	49H	73	i	69H	105
J	4AH	74	j	6AH	106
K	4BH	75	k	6BH	107
L	4CH	76	l	6CH	108
M	4DH	77	m	6DH	109
N	4EH	78	n	6EH	110
O	4FH	79	o	6FH	111
P	50H	80	p	70H	112
Q	51H	81	q	71H	113
R	52H	82	r	72H	114
S	53H	83	s	73H	115
T	54H	84	t	74H	116
U	55H	85	u	75H	117
V	56H	86	v	76H	118
W	57H	87	w	77H	119
X	58H	88	x	78H	120
Y	59H	89	y	79H	121
Z	5AH	90	z	7AH	122
[	5BH	91	{	7BH	123
\	5CH	92 |	7CH	124
]	5DH	93	}	7DH	125
^	5EH	94	~	7EH	126
_	5FH	95	Delete	7FH	127

* * * * * TRS-80 Graphic Characters * * * * *

TRS-80 Graphic Characters are represented by the hexadecimal characters 81H through 0BFH (decimal 129 through 191) and are shown in Appendix E.

* * * * * TRS-80 Space Compression Codes * * * * *

Space compression codes are single bytes that take the place of a number of spaces. In other words, when the TRS-80 encounters these characters in an input stream, it will substitute a number of spaces for that character.

Here is an example:

```
(in Basic)      "A"+CHR$(202)+"B"
(in Hex)        41CA42
```

will expand to:

```
(in print)      "A          B"
(in Hex)        4120202020202020202042
```

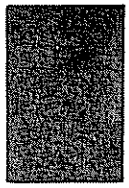
Hex	Decimal	Represents
C2H	194	2 Spaces
C3H	195	3 Spaces
C4H	196	4 Spaces
C5H	197	5 Spaces
C6H	198	6 Spaces
C7H	199	7 Spaces
C8H	200	8 Spaces
C9H	201	9 Spaces
CAH	202	10 Spaces
CBH	203	11 Spaces
CCH	204	12 Spaces
CDH	205	13 Spaces
CEH	206	14 Spaces
CFH	207	15 Spaces
D0H	208	16 Spaces
D1H	209	17 Spaces
D2H	210	18 Spaces
D3H	211	19 Spaces
D4H	212	20 Spaces
D5H	213	21 Spaces
D6H	214	22 Spaces
D7H	215	23 Spaces
D8H	216	24 Spaces
D9H	217	25 Spaces
DAH	218	26 Spaces
DBH	219	27 Spaces

Hex	Decimal	Represents
DCH	220	28 Spaces
DDH	221	29 Spaces
DEH	222	30 Spaces
DFH	223	31 Spaces
E0H	224	32 Spaces
E1H	225	33 Spaces
E2H	226	34 Spaces
E3H	227	35 Spaces
E4H	228	36 Spaces
E5H	229	37 Spaces
E6H	230	38 Spaces
E7H	231	39 Spaces
E8H	232	40 Spaces
E9H	233	41 Spaces
EAH	234	42 Spaces
EBH	235	43 Spaces
ECH	236	44 Spaces
EDH	237	45 Spaces
EEH	238	46 Spaces
EFH	239	47 Spaces
F0H	240	48 Spaces
F1H	241	49 Spaces
F2H	242	50 Spaces
F3H	243	51 Spaces
F4H	244	52 Spaces
F5H	245	53 Spaces
F6H	246	54 Spaces
F7H	247	55 Spaces
F8H	248	56 Spaces
F9H	249	57 Spaces
FAH	250	58 Spaces
FBH	251	59 Spaces
FCH	252	60 Spaces
FDH	253	61 Spaces
FEH	254	62 Spaces
FFH	255	63 Spaces

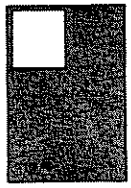
Appendix E: Graphics Table

This appendix was designed to provide a ready visual reference showing each of the TRS-80 Graphic characters. In most tables covering this subject, the light and dark areas have been reversed. We have found this confusing and therefore have shown the pixels (picture elements) in their true representations. For instance, if you want nothing to show, you would want to send a 80H (128 decimal) character to the screen. A completely white block can be placed on the screen with a 0BFH (191 decimal) character.

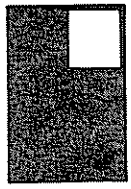
80H 128



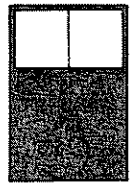
81H 129



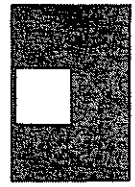
82H 130



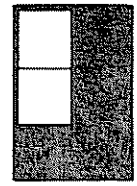
83H 131



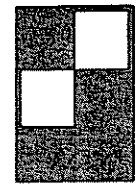
84H 132



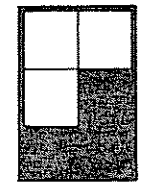
85H 133



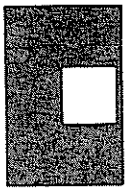
86H 134



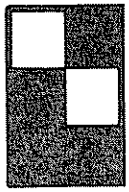
87H 135



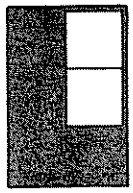
88H 136



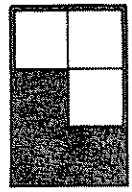
89H 137



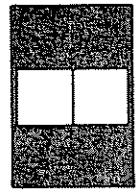
8AH 138



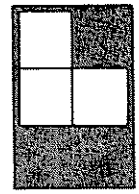
8BH 139



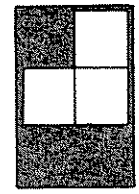
8CH 140



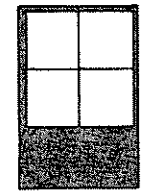
8DH 141



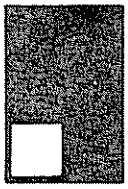
8EH 142



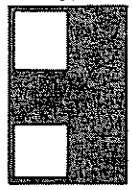
8FH 143



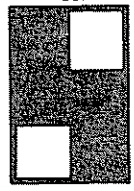
90H 144



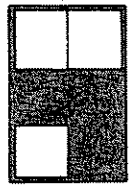
91H 145



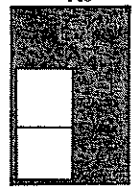
92H 146



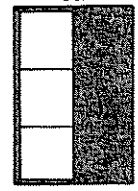
93H 147



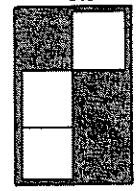
94H 148



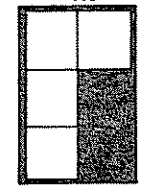
95H 149



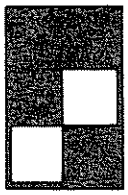
96H 150



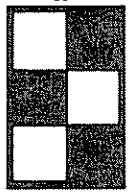
97H 151



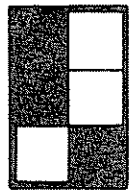
98H 152



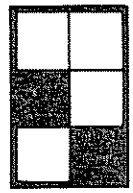
99H 153



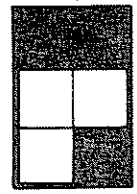
9AH 154



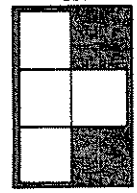
9BH 155



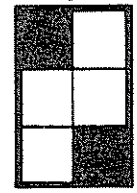
9CH 156



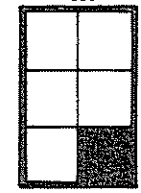
9DH 157



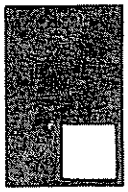
9EH 158



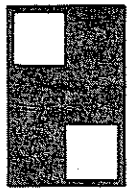
9FH 159



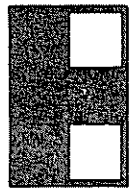
A0H 160



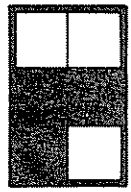
A1H 161



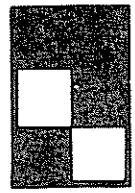
A2H 162



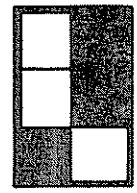
A3H 163



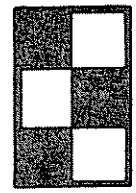
A4H 164



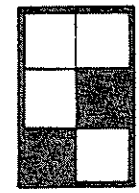
A5H 165



A6H 166



A7H 167



A8H 168

A9H 169

AAH 170

ABH 171

ACH 172

ADH 173

AEH 174

AFH 175

B0H 176

B1H 177

B2H 178

B3H 179

B4H 180

B5H 181

B6H 182

B7H 183

B8H 184

B9H 185

BAH 186

BBH 187

BCH 188

BDH 189

BEH 190

BFH 191

Appendix F: Set/ Reset

```
                01000 ;****************************************
                01010 ;*      SET     RESET     POINT         *
                01020 ;*                                      *
                01030 ;*      Simple Graphics Routines        *
                01040 ;*      by Insiders Software            *
                01050 ;*              Consultants             *
                01060 ;*      PO Box 2441, Dept. SRP          *
                01070 ;*      Springfield, VA   22152         *
                01080 ;****************************************
                01090
1E4A            01100 FCERR   EQU     1E4AH           ;Illegal function call
                01110
                01120 ;********
                01130 ;*      Upon entry by CALL to POINT, SET or RESET (P/S/R)
                01140 ;*      'D'=X coordinate, 'E'=Y coordinate.
                01150 ;*      0<=X<128, 0<=Y<48
                01160 ;********
                01170
7F50            01180         ORG     7F50H
                01190 ;********
                01200 ;*      The POINT routine returns -1 (FFH) if the point
                01210 ;*      is SET, else the value = 00H.
                01220 ;*      The return is in 'A' AND in the INT FPA1 (see
                01230 ;*      volume I)
                01240 ;********
7F50 3E00       01250 POINT   LD      A,0             ;Entry for POINT
7F52 01         01260         DEFB    01              ;Hide the next instr.
                01270                                 ; w/ LD BC
7F53 3E80       01280 SET     LD      A,80H           ;Entry for SET
7F55 01         01290         DEFB    01              ;Hide next instr.
7F56 3E01       01300 RESET   LD      A,01H           ;Entry for RESET
7F58 D5         01310         PUSH    DE              ;Save coords
7F59 F5         01320         PUSH    AF              ;Save command P/S/R
7F5A 7A         01330         LD      A,D             ;LD X Coordinate
7F5B FE80       01340         CP      80H             ;X < 128
7F5D D24A1E     01350         JP      NC,FCERR        ;If not, illegal func.
7F60 F5         01360         PUSH    AF              ;Push X coord.
7F61 7B         01370         LD      A,E             ;Get Y Coord
7F62 FE30       01380         CP      30H             ;Y<48
7F64 D24A1E     01390         JP      NC,FCERR        ;If not <48, illegal func.
```

```
                01400 ;********
                01410 ;*      The next section divides the Y coord. by 3 to get
                01420 ;*      the row ('D') and the remainder ('C')
                01430 ;********
7F67 16FF       01440         LD      D,-1
7F69 14         01450 PSR00   INC     D               ;INC DIV count
7F6A D603       01460         SUB     3               ;Divide by subtraction
7F6C 30FB       01470         JR      NC,PSR00        ;If not neg result,
                01480                                 ; subtract again
7F6E C603       01490         ADD     A,3             ;Restore to positive
                01500                                 ;  (Get remainder)
7F70 4F         01510         LD      C,A             ;Store remainder in 'C'
7F71 F1         01520         POP     AF              ;Get X coord.
7F72 87         01530         ADD     A,A             ;Multiply by two (2)
7F73 5F         01540         LD      E,A             ;Store in 'E'
                01550 ;********
                01560 ;*      Determine the LSB of the position on the screen,
                01570 ;*      the value of which is placed in 'E'
                01580 ;********
7F74 0602       01590         LD      B,2
7F76 7A         01600 PSR01   LD      A,D
7F77 1F         01610         RRA
7F78 57         01620         LD      D,A
7F79 7B         01630         LD      A,E
7F7A 1F         01640         RRA
7F7B 5F         01650         LD      E,A
7F7C 10F8       01660         DJNZ    PSR01
                01670 ;********
                01680 ;*      This section uses the remainder to determine
                01690 ;*      the MSB of the Byte's location in screen memory.
                01700 ;*      The value of the MSB is placed in 'D'
                01710 ;*      The video RAM address is now in 'DE'.
                01720 ;********
7F7E 79         01730         LD      A,C             ;Get remainder
7F7F 8F         01740         ADC     A,A
7F80 3C         01750         INC     A
7F81 47         01760         LD      B,A
7F82 AF         01770         XOR     A
7F83 37         01780         SCF
7F84 8F         01790 PSR02   ADC     A,A
7F85 10FD       01800         DJNZ    PSR02
7F87 4F         01810         LD      C,A
7F88 7A         01820         LD      A,D
7F89 F63C       01830         OR      60              ;Screen line length
7F8B 57         01840         LD      D,A
```

```
                01850 ;********
                01860 ;*      Get character to be manipulated
                01870 ;********
7F8C 1A         01880         LD      A,(DE)
7F8D B7         01890         OR      A
7F8E FA937F     01900         JP      M,PSR03         ;JP if graphics character
7F91 3E80       01910         LD      A,80H           ;Set b7, reset others
7F93 47         01920 PSR03   LD      B,A
7F94 F1         01930         POP     AF              ;Get OP type (P/S/R)
7F95 B7         01940         OR      A
7F96 78         01950         LD      A,B             ;Restore byte
7F97 280F       01960         JR      Z,PSR06         ;JP if POINT
7F99 12         01970         LD      (DE),A          ;Store byte on screen
7F9A FAA57F     01980         JP      M,PSR05         ;JP if SET
7F9D 79         01990         LD      A,C             ;Load bit to reset
7F9E 2F         02000         CPL                     ;All bits 1 except
                02010                                 ; bit to reset
7F9F 4F         02020         LD      C,A
7FA0 1A         02030         LD      A,(DE)          ;Get char again
7FA1 A1         02040         AND     C               ;Reset pixel
7FA2 12         02050 PSR04   LD      (DE),A          ;Put back character
7FA3 D1         02060 PSR04B  POP     DE              ;Restore coords
7FA4 C9         02070         RET                     ;DONE!
7FA5 B1         02080 PSR05   OR      C               ;Set bit
7FA6 18FA       02090         JR      PSR04           ;Finish up
7FA8 A1         02100 PSR06   AND     C               ;Check bit for On/Off
7FA9 C6FF       02110         ADD     A,0FFH          ;If on, RET=-1,
                02120                                 ;  Else RET=0
7FAB 9F         02130         SBC     A,A
7FAC E5         02140         PUSH    HL              ;Save 'HL'
7FAD CD8D09     02150         CALL    098DH           ;Determine sign
7FB0 7D         02160         LD      A,L
7FB1 E1         02170         POP     HL              ;Restore 'HL'
7FB2 18EF       02180         JR      PSR04B
0000            02190         END
00000 Total errors
```

SET/RESET/POINT

FCERR	1E4A	POINT	7F50	PSR00	7F69
PSR01	7F76	PSR02	7F84	PSR03	7F93
PSR04	7FA2	PSR04B	7FA3	PSR05	7FA5
PSR06	7FA8	RESET	7F56	SET	7F53

Appendix G: Lowercase Hardware Modification

Lowercase letters are not supported in an un-modified TRS-80. However, since its introduction, many different modifications have been introduced. One of the most popular ones used to be the "Electric Pencil" modification which added lowercase letters (without descenders on the video) and "Control" key. When Radio Shack decided to produce word processing software, it also needed to provide lowercase letters (who wants to write personal letters ONLY IN UPPERCASE?). For a fee, Radio Shack will install lowercase into your CPU unit which will allow you to display lowercase with one-dot descenders on the screen.

If you have an "old" machine, you may want to spend the money to have the Shack install the modification; this will give you true descenders, and if your seal is still intact, may save you money if the machine fails at a later time. If you have one of the newer machines, REALLY know the in's and out's of computer circuitry, and are not afraid of the consequences of breaking the seal, we are providing this quick LC modification. It should take about 15 minutes to perform.

The lowercase modification on the new machines is easy since the new character generator chip is installed and one of the 2102 RAM chips is in a socket. Proceed as follows:

1) Bend up pins 11 and 12 on a new 2102.

2) Remove Z46-2102 from CPU and piggyback new 2102.

3) Solder all pins together except 11 and 12 - CAREFULLY!
Try not to get solder near the bottom pin end.

4) Solder thin wires to 11 and 12 pins that were bent up. (3-4 inches)

5) Place the 2102s back in the socket; align the notches.

6) Connect pin 11-Z46, to pin 13-Z44.
Use the small hole NE of pin 13.

7) Solder pin 12-Z46, to pin 13-Z27.
You must solder to the pin itself.

8) Cut the trace between Z29 and Z30. The trace is a thin diagonal between Z29 and Z30.

Of course, to get lowercase letters you will have to load a lowercase driver; the assembly language source code for such a driver is listed in Appendix B. Before you open up your unit, remember that it voids all warranties. This modification should not be performed by persons not trained or experienced in servicing SENSITIVE electrical equipment. He that eagerly grabs a soldering gun, whips out the left-over copper wire from the lamp he just put together, and tries to perform the modification will be replacing the $300 board in the CPU.

Appencix H: Printer Driver

```
                    01000 ;****************************************
                    01010 ;*      Parallel Printer Driver         *
                    01020 ;*                                      *
                    01030 ;*      by Insiders Software            *
                    01040 ;*              Consultants, Inc.       *
                    01050 ;*      PO Box 2441, Dept. PRT          *
                    01060 ;*      Springfield, Virginia 22152     *
                    01070 ;****************************************
4026                01080 LPTADR  EQU     4026H           ;Line Printer DCB
4028                01090 LPTLPP  EQU     4028H           ; Lines per page (+3)
4029                01100 LPTLCT  EQU     4029H           ; Line counter (+4)
402A                01110 LPTCPL  EQU     402AH           ; Characters per line (+5)
402B                01120 LPTCCT  EQU     402BH           ; Character counter (+6)
402C                01130 LPTIND  EQU     402CH           ; Line indent (+7)
003B                01140 LPTBYT  EQU     003BH           ;Print a byte
37E8                01150 LPRINT  EQU     37E8H           ;Line printer address
05D1                01160 PSTATU  EQU     05D1H           ;Line printer status ck
                    01170 ;********
F500                01180         ORG     0F500H          ;Anywhere in highmem
F500 2133F5         01190 START   LD      HL,PRTDVR-1     ;Set highmem pointer
F503 224940         01200         LD      (4049H),HL      ;Set HIGH$
F506 22B140         01210         LD      (40B1H),HL      ;Set BASIC highmem ptr
F509 11CEFF         01220         LD      DE,-50          ;Clear string area
F50C 19             01230         ADD     HL,DE
F50D 22A040         01240         LD      (40A0H),HL      ;CLEAR 50
                    01250 ;********
                    01260 ;*      Pointers have been set to protect the driver.
                    01270 ;*      Now, set up the default values in the DCB
                    01280 ;********
0042                01290 LPP     EQU     66              ;Standard page length
0050                01300 CPL     EQU     80              ;Characters/line
0005                01310 IND     EQU     5               ;Line indent
0006                01320 SKPTOF  EQU     6               ;Skip top-of-form flag
                    01330                                 ; if NZ, skip on
                    01340                                 ; eof-skptof
F510 3E42           01350 SETDEF  LD      A,LPP           ;Get lines/page
F512 322840         01360         LD      (LPTLPP),A      ;Set lines/page
F515 3E50           01370         LD      A,CPL           ;Get characters/line
F517 322A40         01380         LD      (LPTCPL),A      ;Set characters/line
F51A 3E05           01390         LD      A,IND           ;Get indent
F51C 322C40         01400         LD      (LPTIND),A      ;Set indent
```

```
F51F AF       01410          XOR     A               ;Zero counters
F520 322940   01420          LD      (LPTLCT),A      ;Zero line counter
F523 322B40   01430          LD      (LPTCCT),A      ;Zero char counter
              01440 ;********
              01450 ;*       Change DCB to reflect new driver
              01460 ;********
F526 2134F5   01470 CHGDCB   LD      HL,PRTDVR       ;New Driver address
F529 222640   01480          LD      (LPTADR),HL     ;Set new driver
F52C 3E0D     01490          LD      A,0DH           ;Print a CR
F52E CD3B00   01500          CALL    LPTBYT
F531 C32D40   01510          JP      402DH           ;Entry to DOS
              01520                                  ;If BASIC, JP 06CCH
              01530 ;********
              01540 ;*       Line Printer Driver
              01550 ;********
F534 79       01560 PRTDVR   LD      A,C             ;Get char to print
F535 B7       01570          OR      A
F536 CAD105   01580          JP      Z,PSTATU        ;Return status if NULL
F539 FE0B     01590          CP      0BH             ;Vert Tab?
F53B 2804     01600          JR      Z,FF            ;Convert VT to FF
F53D FE0C     01610          CP      0CH             ;Form Feed? (FF)
F53F 201F     01620          JR      NZ,CKLF         ;If not FF, ck LF
F541 AF       01630 FF       XOR     A               ;Clear 'A'
F542 DDB603   01640          OR      (IX+3)          ;See if lpp set
F545 C8       01650          RET     Z               ;If no lpp cnt, no out
F546 AF       01660          XOR     A               ;Clear 'A'
F547 DDB606   01670          OR      (IX+6)          ;Middle of line?
F54A 3E0D     01680          LD      A,0DH
F54C C490F5   01690          CALL    NZ,OUTPUT
F54F DD7E03   01700          LD      A,(IX+3)        ;Get lines/page
F552 DD9604   01710          SUB     (IX+4)          ;SUB value in page count
F555 47       01720          LD      B,A             ;Save count
F556 3E0A     01730          LD      A,0AH           ;Output linefeeds
F558 CD7CF5   01740 OUTFF    CALL    OUTA            ;Output 'A' to printer
F55B 10FB     01750          DJNZ    OUTFF           ;Continue for 'B' LFs
F55D C3E1F5   01760          JP      ZERLCT          ;Done. Zero linecount
F560 E67F     01770 CKLF     AND     7FH             ;Reset b7
F562 FE0A     01780          CP      0AH             ;Linefeed?
F564 2007     01790          JR      NZ,CKCR         ;CK for CR if non-LF
F566 CD7CF5   01800 OUTLF    CALL    OUTA            ;Output LF
F569 CDC2F5   01810          CALL    INCLCT          ;Inc line counter
F56C C9       01820          RET                     ;Done w/linefeeds
F56D FE0D     01830 CKCR     CP      0DH             ;CR?
F56F 201F     01840          JR      NZ,OUTPUT
F571 CD7CF5   01850          CALL    OUTA
F574 DD360600 01860          LD      (IX+6),0        ;Zero CCT
F578 CDC2F5   01870          CALL    INCLCT          ;Inc linecounter
F57B C9       01880          RET
```

```
F57C F5          01890 OUTA    PUSH    AF              ;Save char to print
F57D CDD105      01900 OUTA1   CALL    PSTATU
F580 2809        01910         JR      Z,NOABRT        ;If ready, output
F582 3A4038      01920         LD      A,(3840H)       ;Get ENTER/CLR/BRK
F585 E606        01930         AND     6               ;CLR/BRK?
F587 28F4        01940         JR      Z,OUTA1         ;If no pressed, loop
F589 F1          01950 ABORT   POP     AF
F58A C9          01960         RET                     ;Ret as if char sent
                 01970                                 ; to prevent lockup
F58B F1          01980 NOABRT  POP     AF
F58C 32E837      01990         LD      (LPRINT),A      ;output to printer
F58F C9          02000         RET
F590 FE20        02010 OUTPUT  CP      20H             ;Output a byte
F592 3004        02020         JR      NC,NOTCTL       ;JP if non-control
F594 CD7CF5      02030         CALL    OUTA
F597 C9          02040         RET
F598 4F          02050 NOTCTL  LD      C,A             ;Save char to print
F599 DD7E06      02060         LD      A,(IX+6)        ;Get char count
F59C B7          02070         OR      A               ;Non-zero?
F59D 2010        02080         JR      NZ,ALRDYN       ;JP if already indent
F59F DD4607      02090         LD      B,(IX+7)        ;Get indent
F5A2 3E20        02100         LD      A,20H           ;Spaces
F5A4 CD7CF5      02110 OUTIND  CALL    OUTA            ;Output indent
F5A7 10FB        02120         DJNZ    OUTIND
F5A9 DD7E07      02130         LD      A,(IX+7)        ;Get indent
F5AC DD7706      02140         LD      (IX+6),A        ;Save CCT
F5AF 79          02150 ALRDYN  LD      A,C             ;Get character
F5B0 CD7CF5      02160         CALL    OUTA
F5B3 DD3406      02170         INC     (IX+6)          ;Inc CCT
F5B6 DD7E05      02180         LD      A,(IX+5)        ;Get CPL
F5B9 DDBE06      02190         CP      (IX+6)
F5BC C0          02200         RET     NZ
F5BD 3E0D        02210         LD      A,0DH           ;Carriage return
F5BF C36DF5      02220         JP      CKCR
F5C2 DD3404      02230 INCLCT  INC     (IX+4)          ;INC Line Counter
F5C5 DD7E03      02240         LD      A,(IX+3)        ;Get LPP
F5C8 DDBE04      02250         CP      (IX+4)
F5CB CCE1F5      02260         CALL    Z,ZERLCT
F5CE D606        02270         SUB     SKPTOF          ;Skip TOF count
F5D0 DDBE04      02280         CP      (IX+4)
F5D3 C0          02290         RET     NZ              ;Return if not at bottom
F5D4 0606        02300         LD      B,SKPTOF
F5D6 3E0A        02310         LD      A,0AH           ;Output to TOF
F5D8 CD7CF5      02320 OUTTOF  CALL    OUTA
F5DB 10FB        02330         DJNZ    OUTTOF
F5DD CDE1F5      02340         CALL    ZERLCT
F5E0 C9          02350         RET                     ;At tof
F5E1 DD360400    02360 ZERLCT  LD      (IX+4),0        ;Zero line counter
F5E5 C9          02370         RET
F500             02380         END     START
```

```
00000 Total errors

ABORT    F589 ALRDYN   F5AF CHGDCB   F526
CKCR     F56D CKLF     F560 CPL      0050
FF       F541 INCLCT   F5C2 IND      0005
LPP      0042 LPRINT   37E8 LPTADR   4026
LPTBYT   003B LPTCCT   402B LPTCPL   402A
LPTIND   402C LPTLCT   4029 LPTLPP   4028
NOABRT   F58B NOTCTL   F598 OUTA     F57C
OUTA1    F57D OUTFF    F558 OUTIND   F5A4
OUTLF    F566 OUTPUT   F590 OUTTOF   F5D8
PRTDVR   F534 PSTATU   05D1 SETDEF   F510
SKPTOF   0006 START    F500 ZERLCT   F5E1
```

Appendix I: Tables

Hexadecimal Columns

	6	5	4	3	2	1
HEX	DEC	DEC	DEC	DEC	DEC	DEC
1	1,048,576	65,536	4,096	256	16	1
2	2,097,152	131,072	8,192	512	32	2
3	3,145,728	196,608	12,288	768	48	3
4	4,194,304	262,144	16,384	1024	64	4
5	5,242,880	327,680	20,480	1280	80	5
6	6,291,456	393,216	24,576	1536	96	6
7	7,340,032	458,752	28,672	1792	112	7
8	8,388,608	524,288	32,768	2048	128	8
9	9,437,184	589,824	36,864	2304	144	9
A	10,485,760	655,360	40,960	2560	160	10
B	11,534,336	720,896	45,056	2816	176	11
C	12,582,912	786,432	49,152	3072	192	12
D	13,631,488	851,968	53,248	3328	208	13
E	14,680,064	917,504	57,344	3584	224	14
F	15,728,640	983,040	61,440	3840	240	15

Powers of 2

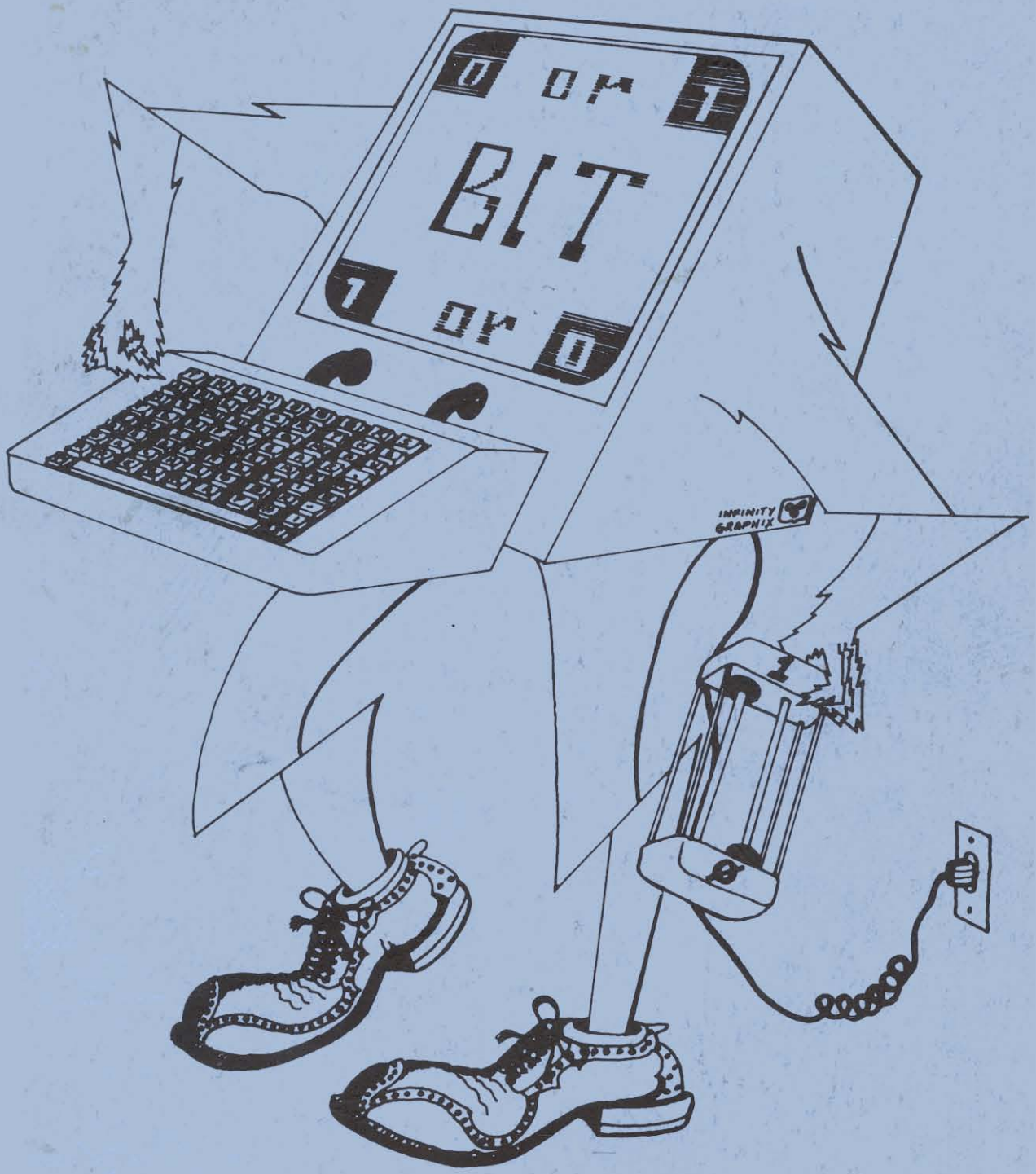
2^8	=	256
2^9	=	512
2^{10}	=	1,024
2^{11}	=	2,048
2^{12}	=	4,096
2^{13}	=	8,192
2^{14}	=	16,384
2^{15}	=	32,768
2^{16}	=	65,536
2^{17}	=	131,072
2^{18}	=	262,144
2^{19}	=	524,288
2^{20}	=	1,048,576
2^{21}	=	2,097,152
2^{22}	=	4,194,304
2^{23}	=	8,388,608
2^{24}	=	16,777,216

Conversion Algorithm

$$2^N = 16^{N/4}$$

Powers of 16

1	=	16^0
16	=	16^1
256	=	16^2
4,096	=	16^3
65,536	=	16^4
1,048,576	=	16^5
16,777,216	=	16^6


0 or 1
BIT
1 or 0
INFINITY GRAPHIX