SYSTEM PROGRAMMER'S GUIDE

for the

TRS-80® Model 4/4P

using

Montezuma Micro CP/M® 2.2
Version 2.2x
# TABLE OF CONTENTS

1. INTRODUCTION ......................................................... 1
2. THE SYSTEM PARAMETER BLOCK ..................................... 3
3. I/O USING THE IOBYTE ................................................ 5
4. THE KEYBOARD DRIVER ................................................ 9
5. THE VIDEO DISPLAY DRIVER .......................................... 11
6. THE PARALLEL PRINTER DRIVER .................................... 13
7. THE SERIAL PORT DRIVER ........................................... 15
8. THE MEMORY DISK DRIVE ............................................ 17
9. DISK I/O ................................................................. 19
   9.1. The Floppy Disk Driver ........................................ 19
   9.2. DPB Extensions .................................................. 19
   9.3. DCB Definitions ................................................ 21
   9.4. Using the Disk Driver ......................................... 22
   9.5. EXBIOS - EXTending the BIOS ............................... 23
   9.6. The Hard Disk Driver .......................................... 23
10. CP/M BOOTS .......................................................... 25
    10.1. The Cold Boot ................................................ 25
    10.2. The Warm Boot ............................................... 25
11. PITFALLS AND TRAPS ............................................... 27
    11.1. INTERRUPTS .................................................. 27
    11.2. FEATURES UNIQUE TO THE Z80 ............................ 27
    11.3. RAM USAGE .................................................... 28
12. INDEX ........................................................................ 29
13. THE LISTING .................................................................. 31
1. INTRODUCTION

All of the information in this manual is copyrighted by Montezuma Micro, including the source code listings. The purpose of this manual is to provide you with the information necessary to modify the BIOS of your own copy of CP/M®. Much like the service manual for a car, this document will show you how the BIOS works, and how you can use your own options with it. It will NOT teach you how to use CP/M and it will NOT teach you how to write assembly language programs for use with CP/M. If you bought it for that purpose you will be frustrated and confused. Let us state from the outset that this manual is for experienced programmers! We absolutely cannot and will not provide any telephone or written support for any modifications made to CP/M. In short, you are ON YOUR OWN and if the information you need can't be found in this manual it simply is not available from us.

Furthermore this manual is applicable to Montezuma Micro CP/M BIOS version 2.2x, where x is the patch level. When future versions of our CP/M are released the listings and possibly some of the information will no longer be valid. We make no promises of any kind as to the availability of a similar manual for future versions, and offer no "upgrades" of any kind on this document.

The BIOS listing which accompanies this manual was created using the 2500 A.D. Z80 Macro Assembler. This assembler uses Z80 mnemonics, unlike the ASM assembler provided with CP/M which uses only 8080 codes. Without apology the author admits to a strong bias for the Zilog Z80 mnemonics, and a strong distaste for the Intel 8080 mnemonics. Only Zilog mnemonics will be used in this manual. The 2500 A.D. assembler is available from Montezuma Micro, and is highly recommended for Z80 programming.

Well, now that we have the preliminaries out of the way let's proceed!
2. THE SYSTEM PARAMETER BLOCK

MOVCPM is a very handy utility which makes it possible to change the size of CP/M so as to reserve space at the top of memory. Unfortunately this creates a major headache for the programmer who wants to write utilities to run under CP/M, since it is not possible to know exactly where in memory each individual copy of CP/M resides. Furthermore some parts of the BIOS may be relocated in the event of an update, necessitating the update of all related utility programs.

To solve these problems we have collected all of the “need to know” information into a section of memory called the System Parameter Block (SPB). The relative location of this block within the BIOS is guaranteed not to change from one version of the BIOS to the next. Further, any additions to it will be made to the end so that relative offsets within the SPB will be good in future versions.

Location of the SPB within CP/M is very simple. It is always 48 bytes (0030H) past the Warm Boot vector in the BIOS. Since the address of the Warm Boot vector always follows the JMP instruction at memory location 0 the SPB can be found using this simple routine:

LD HL,(0001H) ;Get Warm Boot vector address
LD BC,0030H ;Set up offset to SPB
ADD HL,BC ;HL now points to SPB

The remainder of this chapter will deal with the various fields of the SPB. All offsets are given in decimal. Conversion to hex is left as an exercise to the reader.

**Offset 0**
The first field of the SPB is a single byte which contains the standard system IOBYTE value set by the CONFIG utility. At each warm boot the contents of this byte are copied to location 3.

**Offset 1**
Acting as a flag, the contents of this byte tell the BIOS whether to display the CP/M banner after booting. Any non-zero value will cause the banner to be displayed, while zero suppresses it.

**Offset 2**
In this byte is stored the total number of disk drives, as set by CONFIG. It is never used by the BIOS, but may be of use to external utilities.

**Offset 3**
The current version of the CP/M BIOS is stored here as two BCD digits packed in a single byte. The first digit is the release number, which changes only upon a complete rewrite. The second is the revision level, which changes upon reassembly of the BIOS. In a fit of optimism the BIOS programmer set this byte to 20H, meaning 2.00. By the time release 2 was ready the revision level had crept up to 2, but the byte was left at 20H. Thus a value of 20H in this byte should be treated as being synonymous with 22H.

**Offset 4**
Access to disk drives in the BIOS is done using a data structure known as the Disk Parameter Header. It is discussed fully in the manual provided with your CP/M. To allow for the maximum 16 drives possible within CP/M (A: through P:) we have built a 32 byte table of DPH addresses within the BIOS. Whenever a drive is selected via BIOS call XX1BH its corresponding address in this table is
returned. Unused entries are set to 0000H. The two byte address contained in this offset of the SPB is the actual base address of the DPH table, i.e. the address of the address of the DPH for A:. By using relative offsets to this address utility programs may add logical drives to the DPH table. Simply store the address of the DPH of your logical drive at the corresponding entry in the table. Use extreme caution with drive M:, however. The BIOS disk read/write routines test for drive M: and transfer control to special driver coding for that case. Storing the DPH for any other drive in the M: slot will cause problems.

At boot time the DPH table is filled with zeroes and the slots for A:, B:, C:, and D: are filled with the addresses of the four DPHs resident within the BIOS. If the system has 128k of RAM the DPH for drive M: is also added to the table. When the system is booted from a hard disk a patch in the disk boot causes this table to be overwritten with the configuration specified when the hard disk driver was installed.

**Offsets 6, 8, 10, and 12**

These four offsets contain the two-byte addresses of the disk Device Control Blocks (DCB) for each of four possible floppy disk drives 0 through 3 respectively. The disk DCB, which will be discussed in depth in the DISK I/O section, is used to access a particular floppy drive and contains all the physical characteristics of that drive.

**Offset 14**

At this offset in the SPB is a two-byte address which points to the base of a table of device driver addresses. This table is used by the BIOS for all I/O except for disk and has been designed to simplify the installation of custom drivers. See I/O USING THE IOBYTE for full details.

**Offset 16**

The base address of the Keyboard Device Control Block (DCB) is found at this offset. See THE KEYBOARD DRIVER for information regarding the Keyboard DCB.

**Offset 18**

Here is the base address of the Video Display DCB. This data structure is fully explained in THE VIDEO DISPLAY DRIVER. One item of interest in this DCB is the current cursor location.

**Offset 20**

The base address of the Parallel Printer Port DCB is stored here. For a full explanation of the DCB see THE PARALLEL PRINTER DRIVER.

**Offset 22**

This is the end of the SPB, at least for now. Any extensions made in future versions of our CP/M BIOS will be made starting at offset 24, thereby retaining compatibility with programs written for earlier versions. You are encouraged to use this structure whenever you must “peek”, “poke”, or otherwise farkle with the BIOS. Doing so will make your life easier and could help to remove unsightly warts!
One of the optional features of CP/M 2.2 that we have implemented is the IOBYTE. The IOBYTE is a set of four two-bit (literally!) fields that can be manipulated to change the assignment of real devices (Keyboard, Serial Port, etc.) to logical devices (CONsole, Line PrinTer, etc.). By convention it resides in memory at location 3 (0003H for hard-core hexaphiles) and is structured like this:

<table>
<thead>
<tr>
<th>Logical Device ---</th>
<th>LST:</th>
<th>PUN:</th>
<th>RDR:</th>
<th>CON:</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bits Used ---</td>
<td>7, 6</td>
<td>5, 4</td>
<td>3, 2</td>
<td>1, 0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Decimal</th>
<th>Binary</th>
<th>Physical Device</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>00</td>
<td>TTY: TTY: TTY: TTY:</td>
</tr>
<tr>
<td>1</td>
<td>01</td>
<td>CRT: PTR: CRT:</td>
</tr>
<tr>
<td>2</td>
<td>10</td>
<td>LPT: HIM: UR1: BAT:</td>
</tr>
<tr>
<td>3</td>
<td>11</td>
<td>UL1: UP2: UR2: UC1:</td>
</tr>
</tbody>
</table>

The logical devices referenced above are as follows:

**LST:** The List device, typically used in CP/M for hardcopy, is output only. Usually it is assigned to a printer.

**PUN:** The Punch device reveals the origins of CP/M, back when a paper tape reader/punch was the norm for microcomputers. While such hardware is now relegated mainly to museums and junk yards, we still have this output (only) device to use as we see fit.

**RDR:** Like PUN: the Reader device is a throwback to those golden days of 1k RAM boards and Tiny BASIC. Use it as you wish for input (only) operations.

**CON:** Perhaps the most important device from an operational standpoint is the CONsole. It is the device from which CP/M gets its commands and to which it sends its output. You must be very careful in making assignments to this device, since mistakes can cause you to be unable to communicate with CP/M at all!

Now let’s look at the case of physical devices, as defined within our CP/M:

**TTY:** Another relic from the Neanderthal age of computing is the TeleTYPE. This was a large, noisy machine consisting of a keyboard and a printer. It was able to send and receive data at the blazing speed of 10 characters per second. We have assigned this device to the Serial Port of the Model 4, since the original TTY was serial and you could actually connect one to this device if you really wanted to.

**CRT:** CRT stands for Cathode Ray Tube and as come to stand for just about any kind of terminal using video output. We have defined the CRT to be the Keyboard of the Model 4 on input and the Video Display on output.

**PTP:** Since there is no Paper Tape Punch on the Model 4 we have assigned this device to the Video Display. You could, of course attach a true paper tape punch to that port, but if you think about this sort of thing often you really ought to get professional help.
PTR: The Paper Tape Reader falls in the same category as the PTP. Since there (thankfully) is no such device on the Model 4 we use the Keyboard.

LPT: This device is the Line PrinTer, which could be either a serial or a parallel device. Since we already have TTY: for serial output LPT: is assigned to the parallel port. It is, by nature, output only.

UP1:UP2: UP1 and UP2 are user-defined punch devices. The BIOS maps both of these devices to a null driver which does nothing, but is never busy and will not hang up the system if used. Both of these devices are available for user-written drivers.

UR1:UR2: Like UP1 and 2 the UR1 and UR2 drivers are user-defined, but these are reader devices. The BIOS, as shipped, has both devices assigned to a null driver which provides only the end-of-file character Z as input, but will not hang up the system. Both are available for use with user-written drivers.

UL1: The UL1 device is a user-defined line printer. It is, by nature, output only and is assignable only to the LST: logical device. Interfacing of special output devices, e.g. a plotter, can be done using this device. As shipped this device is assigned to a null driver.

UC1:UC1 is the user-defined console device. If implemented it must provide both input and output capability. Unlike other user-defined devices, this one is preassigned to the Keyboard driver for input and the Video Display driver for output. This was done so that there would be no nasty loss of control should the device be accidentally assigned.

BAT: In olden times when computers were slow and I/O devices were even slower it was common to set up a “batch” of instructions for the machine, start it up, and leave for a two-week vacation. We have implemented the BATch device faithful to its original use. When input is taken from BAT: it actually comes from whatever is assigned to the RDR: device. Output to BAT: actually goes to the LST: device. As you can see BAT: is not a physical device, but merely a switch to change the assignment of CON:.

Now that we have seen just what devices are available, how do we go about actually installing a driver for one? Obviously you must first write the driver. What it does is up to you, but there are some conventions that must be followed if the driver is to work properly in CP/M. At minimum it must provide the following four functions:

**Input Status**
No parameters are required. If the device has input available it should return 0FFH in the A register, otherwise return 0.

**Input Data**
A byte of input is read from the device and returned in the A register. If no input is available this routine must wait until it is.

**Output Data**
The C register contains the byte of data to be output to the device. No value is returned to the caller after output.
Output Busy
The output device is tested for a busy condition. If it is busy a value of 0 is returned in the A register. Ready is indicated by a return of 0FFH in the A register.

Of course your device may not actually need all of the above routines, especially if it is input-only or output-only. Unused routines should be replaced with null driver code, so as to prevent problems should a calling routine do something stupid. A sample null device driver can be found in the BIOS listing.

Installation of a driver is not difficult. First you need to find a place in RAM to hold the driver. If it replaces an existing CP/M driver you may want to load it over the existing driver. Be sure that you don’t overwrite other code if you do this! The safest method is to locate the driver either above or below CP/M. To load it above CP/M use MOVCPM to create a smaller system. This is preferable to loading it below CP/M, since that method makes the driver vulnerable to being destroyed by programs which use all of available memory. Once you have located the driver in RAM it is necessary only to install the addresses of the above four routines in the proper place within the Device Driver Address Table in the BIOS. The base address of this table can be obtained from offset 14 in the System Parameter Block (SPB) discussed previously. Each entry in this table consists of four two-byte addresses arranged in the following order:

Address of Input Status routine
Address of Input Data routine
Address of Output Data routine
Address of Output Busy routine

Each device holds a particular entry in the table. The devices and their respective offsets are as follows:

<table>
<thead>
<tr>
<th>Device</th>
<th>Offset</th>
<th>Device</th>
<th>Offset</th>
<th>Device</th>
<th>Offset</th>
</tr>
</thead>
<tbody>
<tr>
<td>TTY:</td>
<td>0</td>
<td>CRT:</td>
<td>8</td>
<td>UC1:</td>
<td>16</td>
</tr>
<tr>
<td>LPT:</td>
<td>24</td>
<td>UL1:</td>
<td>32</td>
<td>PTR:</td>
<td>40</td>
</tr>
<tr>
<td>UR1:</td>
<td>48</td>
<td>UR2:</td>
<td>56</td>
<td>PTP:</td>
<td>64</td>
</tr>
<tr>
<td>UP1:</td>
<td>72</td>
<td>UP2:</td>
<td>80</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

These offsets will be held constant in future versions of CP/M. It is unlikely that any new devices will be added, but should that happen the new devices would be added after all existing ones.
4. THE KEYBOARD DRIVER

The console device CON: is CP/M’s main source of input other than disk. Most of the time the Model 4 keyboard serves as the primary input device for CON:.

At first glance the keyboard looks like a fairly simple device to interface with. Just scan the rows and see which key is being pressed, then return the ASCII value for that key. Unfortunately the problem is not that simple since you have to contend with SHIFT, CTRL, rollover, repeating keys, function keys, debounce, and a whole lot of other annoying stuff. Our methods for dealing with this chaos may be found in the keyboard driver portion of the BIOS listing. The purpose of this section of the manual is not to explain how the driver works, but rather to discuss the Keyboard Device Control Block (DCB).

The various fields of the Keyboard DCB will be discussed using the field name as shown on the BIOS listing, as well as the decimal offset that the field has from the base of the DCB.

**KBDBUF - Offset 0**

When the Keyboard driver is called to check for pending input it must scan the keyboard to see if a key is depressed. Since the status routine does not return the actual key value, but only a flag, it is possible that the key will no longer be depressed when the input routine is called to read it. To prevent that from happening any key that is found will be stored in this buffer. On future calls to the status routine if anything is in the buffer a 0FFH will be returned without doing another keyboard scan. Likewise the input routine always checks the buffer before scanning the keyboard for input. It is important to note that his buffer is for only one character and is not in any sense a type-ahead buffer.

**KBDFKP - Offset 1**

The function keys can be set up to deliver from 0 to 8 characters when pushed. Since only one character is transmitted on each call to the driver a way is needed for the driver to “remember” what the rest of the characters are. This is done using this two-byte pointer, which contains the address of the next character to be used as input.

On every call to the keyboard input routine KBDFKP is checked for a non-zero value. If it is found to be non-zero then a character is loaded from the address in KBDFKP. As long as the loaded character is not zero it is returned to the calling program just as though it had been typed. When a zero value is read KBDFKP is set to zero and the driver resumes keyboard scanning.

Although the function keys are located within the Keyboard Driver in the BIOS, it is possible to set KBDFKP to point to strings of “key input” in other parts of memory. This is exactly the technique used by Montes Window to return Calculator results as keyboard input. There is no limit to the length of the pseudo keyboard input, but the last byte must be 0.

**KBDHST - Offset 3**

This 8 byte field is used to hold the results of previous scans of each of the 8 keyboard row lines. The information is used to lock out keys that were depressed on previous calls to the driver. Since these fields are necessary for the correct implementation of rollover and repeat it is strongly recommended that you don’t mess with them!
KBDPKR - Offset 11
KBDPKI - Offset 12
KBDDLY - Offset 14
KBDRPT - Offset 16

These fields are all used in controlling the automatic repeat of a key held down more than a few seconds. Tweaking and other perverse manipulation is likely to have bad results!

KBDCLF - Offset 18

The Caps Lock Flag is used by the driver to remember whether the keyboard is locked into all upper case or not. Only bit 0 of this byte is actually used but care should be used to keep the other 7 bits set to 0. A value of 0 in this byte indicates that the keyboard is operating in normal upper/lower case, while a value of 1 says that it is locked into all-caps. This flag may be changed so long as the only values used are 0 and 1.

KBDCOD - Offset 19

All of the alpha keys are decoded “on the fly” using the scan information to generate ASCII codes immediately. This technique is not easily applied to the numeric and special function keys, so those are decoded using a table. The Keyboard Decode Table is divided into three entries of 24 bytes each. Each entry defines the following keys in this order:

```
0 1 2 3 4 5 6 7
8 9 ; , . /
ENTER CLEAR BREAK UP DOWN LEFT RIGHT SPACE
```

The first entry defines the above keys used alone, i.e. without either SHIFT key or the CTRL key. Definitions for SHIFT in combination with these keys are in the second entry, and the last entry is for CTRL definitions. Obviously you can change these key definitions to virtually anything you want. We do offer a few recommendations. The ENTER key should produce a carriage return whether SHIFTed, CTRLed, or used alone. That’s why our utilities don’t allow it to be changed. It would probably also be a good idea for the normal and SHIFTed keys to produce the character inscribed on the keytop. Beyond that, have fun!

KBFKD - Offset 91

The Function Key Definition table contains the strings to be issued when any of the nine possible function key combinations is selected. This table has nine entries of nine bytes each. The first three correspond to unshifted F1, F2, and F3, while the next three are used for SHIFT/ F1, SHIFT/ F2, and SHIFT/ F3. CTRL/ F1, CTRL/ F2, and CTRL/ F3 are defined in the last three entries. Each definition string may be from 0 to 8 characters long, but the last byte in the string MUST be 0. That’s the signal for the Keyboard driver to stop.

This concludes the Keyboard DCB. Since this data structure is used only by the BIOS Keyboard driver it need not be present for Keyboard drivers that you may write yourself. However the last two fields (KBDCOD and KBFKD) must be present at those offsets if the CONFIG utility is to be used with your driver. Field KBFKFP must be present for Monte’s Window to work, and that product may not work at all if your Keyboard driver is not physically located in the BIOS memory space.
5. THE VIDEO DISPLAY DRIVER

The Video Display driver uses the Model 4 memory-mapped Video Display to emulate a Lear-Siegl
ADM-3A terminal. Most CP/M compatible software expects the CON: device to be an ASCII terminal
and the ADM-3A is one of the more common terminals in use, so this arrangement works well. At all
times except when it is actually being updated the Video Display RAM is kept switched out of the
memory map. This allows BIOS code to occupy the same space and reduces the overall memory
overhead on the system.

Operation of the Video Display Driver is uncomplicated and should be easy to follow in the BIOS
listing. The main purpose of this section is to explain the Video Display Device Control Block (DCB).
This discussion will use the field names from the listing and their offsets from the base of the DCB.
As mentioned earlier the address of the Video Display DCB can be obtained from the System Param-
eter Block (SPB).

VDDCHR - Offset 0
Depending on the state of the Video Display the cursor character may be either an inverse-video form
of the character under it or a wedge-shaped character. This field keeps the character under the cursor
so that the driver may easily replace it when moving the cursor.

VDDROW - Offset 1
In this byte is stored the current cursor row address, which will have a value varying from 0 to 23.
The top line of the screen is row 0. Programs may interrogate this field to “find” the cursor on the
screen, but should not write to it as a means of moving the cursor. If you do the Video Display driver
will not be able to erase the old cursor and you will soon have a display full of cursors with only one
of them being real.

VDDCOL - Offset 2
The current cursor column address is stored in this byte. It will range from 0 to 79 with 0 being the
leftmost column on the screen. As with VDDROW this field should be considered “read only”.

VDDINV - Offset 3
Data may be displayed in normal video (white characters on black background) or inverse video (black
characters on white background). The current video mode is kept in bit 7 of this byte. A value of 0
means normal video, while 1 (80 hex) means inverse. No other value should ever be put in this byte
except 0 and 80H.

VDDESC - Offset 4
VDDESX - Offset 5
These fields are used in processing the ESCape sequence used for cursor positioning. They provide
no useful information to external programs and should never be modified.

VDCXAT - Offset 6
This table is a series of 32 byte entries that correspond to the control codes from 0 to 31 (00H to
1FH). Originally the table consisted of two-byte addresses of service routines to process the various
control codes. During the final stages of implementing release 2.00 of CP/M, though, the bytes got
very scarce and this table was converted to 32 one-byte offsets in a desperate attempt to recover space.
This makes it somewhat unstable for future versions, since it cannot span a 256 byte RAM boundary,
and it should not be counted on in subsequent releases.
The reason for using the table was so that any unused video control codes could be assigned as duplicates of existing codes to possibly provide compatibility with other computers. In particular the Kaypro series can be closely emulated in this way. When using this table the offsets which are defined should be considered read-only, while the others may be modified as needed. Don't bank on this table working the same in future generations.

This concludes the Video Display DCB. You may replace the entire Video Display driver with one of your own design, but be advised that Monte's Window may not function correctly if the driver is not located in BIOS RAM space. Since part of the BIOS code resides in the Video RAM space you must also switch the Video RAM in and out of the map as the existing driver does.
6. THE PARALLEL PRINTER DRIVER

The Model 4 computer is equipped with a Centronics-compatible parallel printer port. Our CP/M BIOS provides a driver for this port with some options that are contained in a Device Control Block (DCB). Operation of the driver is quite simple and should be apparent from the BIOS listing. The purpose of this section is to describe the DCB used for this port. Each field of the DCB and its offset follows.

**PPDPRV - Offset 0**
The character last output to the printer is kept here. This is needed whenever a linefeed is output, since we may not wish to output it at all depending on what went before. Manipulating this byte probably serves no useful purpose to an external program.

**PPDOPT - Offset 1**
All options set by the CONFIG utility are stored here. At present there are only two options. Bit 0, if set to 1, indicates that linefeeds which follow carriage returns are not to be output. Setting this bit to 0 causes all linefeeds to be output normally. Bit 1 is used to control the simulation of the formfeed character (0CH). For printers which do not recognize that code setting this bit to 1 will cause it to be simulated by repeated linefeeds. To do this the page length in lines must be known, along with a count of how many lines have been printed already. That information is kept in the following two fields. Bits 2 through 7 of this byte are not, as yet, used but are reserved for future use.

**PPDLCT - Offset 2**
This contains the count of the number of lines left to print on the current page. It is decremented each time a linefeed is sent to the driver, even if the linefeed was not actually performed. Unless bit 1 of PPDOPT is set this field serves no purpose. It is reset to the value in the next field after each Warm Boot.

**PPDPGL - Offset 3**
The number of lines on one page is kept in this byte. Unless changed by CONFIG the default is 66, which is a standard eleven inch page at six lines per inch. At Warm Boot and end of page time this value is loaded into PPDLCT.

There are no more fields in the Parallel Printer DCB. If you choose to replace this driver you may choose to use the existing DCB just so you can have access to the CONFIG settings.
7. THE SERIAL PORT DRIVER

The Serial Port on the Model 4 provides a standard RS-232C interface for external devices. Unfortunately the only thing standard about RS-232C seems to be which lines are used for data and ground. All others tend to change from one manufacturer to another. We have attempted to create a driver that will deal with as many different configurations as possible, but it is by no means comprehensive. In this section we will describe how the DCB is used. Operation of the driver, which is quite short, should be apparent from the BIOS listing. As with all the other drivers in this manual, each field is listed by name and offset.

SPDINT - Offset 0
This three-byte field is actually an executable Z-80 instruction. It contains a JuMP (opcode C3H) to a routine which will initialize the Serial Port and return. We had to provide for this capability since a change in Serial Port parameters almost always requires that the port be reinitialized. At this time this routine is used by CONFIG and MODEM7 (version 7.31). The initialization routine must end with a RET instruction (C9H).

SPDOPT - Offset 3
All of the options for the serial port driver are contained in this byte as bit flags. Bit 0, if set, tells the driver not to output data until the CTS (Clear To Send) line goes true. Similarly bit 1 is used to suspend output until DSR (Data Set Ready) becomes true. Bits 2-7 are reserved, but not in use at this time. The CTS and DSR lines of the Serial Port have presented somewhat of a problem for many users. On output the RTS (Request To Send) and DTR (Data Terminal Ready) lines must be inverted by the software. Since CTS and DSR are essentially the same signals coming from the other end of the RS-232 link we also inverted them before testing. This caused problems for many users, especially those with serial printers that use the lines for handshaking. To correct the problem we came out with version 2.22. The only change in this version from 2.21 was the removal of the XOR 80H following SPBSY in the Serial Port driver, as well as the XOR 40H following SPBSY1. These two instructions were removed by storing two bytes of zero (the NOP instruction) over the XORs.

SPBBDR - Offset 4
A single byte containing the code to be output to the baud rate generator is stored here. It is actually output when SPDINT is called. See the Model 4 Technical Reference Manual (any version) for a list of the codes.

SPDCFG - Offset 5
When SPDINT is called the byte in this field is output to the UART control register to configure it. As with SPBBDR you can get the codes from the Model 4 Technical Reference Manual.

That’s all there is to the Serial Port DCB. Although we do have the hardware handshake problem addressed with this driver, it does not handle the common XON-XOFF software protocol used by many devices. In any future revision this option would have a high priority. It was omitted due to RAM constraints.
8. THE MEMORY DISK DRIVE

One of the features added in release 2 of the CP/M BIOS was the enabling of the Memory Disk (Drive M:) automatically at boot time. In release 1 the drivers for the Memory Disk were not part of the BIOS, but were loaded externally below the CCP. This was a messy implementation since the CCP then had to remain resident, resulting in a smaller TPA, and the system configuration could not be saved using CONFIG. By moving the drivers into the BIOS both problems were solved.

During a CP/M Cold Boot (RESET button) a test is made to see if the system has 128k of memory installed. If it does the 64k expansion RAM is filled with E5H bytes to make it appear to CP/M as a blank disk, and the DPH for drive M: is installed in the DPH table. When no extra RAM is present the entry in the DPH table for drive M: is simply left zero-filled.

Several users have requested the ability to leave drive M: in an uninitialized state, so that data could be preserved from one boot to another, or even from one DOS to another. Of course this can only happen if power has been constantly turned on between boots. By patching a RET instruction (C9H) into BOOT2 in the BIOS you can prevent drive M: from being initialized. However this leaves no way to ever get it initialized, which must be done when the system is first powered up. In future revisions we will probably provide for drive M: to be preserved if we can establish that it contains a flag showing that it was set up correctly.

Other users have asked that drive M: be permanently disabled and not initialized at all. This can be done simply by changing the JR NZ (opcode 20H) at location EA8BH on the BIOS listing to JR (opcode 18H). Here, too, we will consider implementing this capability on future versions.

Although the actual drivers for reading and writing to drive M: are quite short they may not be readily understandable. The problem is the “magic” that occurs when the expansion RAM is switched in and out of the memory map. We used a special 128 byte buffer in the BIOS to hold records on their way to or from drive M:. After calculating the expansion bank and RAM address of the desired “sector” the required bank is switched into the map at location O000H. The transfer is made using the BIOS buffer, and the bank is switched back out. Because the first 32k is involved in this the BIOS can NEVER reside in any part below address 8000H. In other words it must always be in the top 32k of RAM.

This section on the RAM disk has been included mainly for the curious. It is not recommended that any tinkering be done on this driver. Those who choose to do so anyway should be warned that some products, notably Monte’s Window, may fail miserably if the RAM disk is farkled.
9. DISK I/O

The code to control the disk drives is one of the main parts of the BIOS. CP/M is a disk-based operating system and makes frequent use of disk drives. Floppy disks are standard on the Model 4/4P, and you may also optionally attach a hard disk drive. Code for the floppy disk is an integral part of the BIOS, but hard disk drive require the installation of a separate driver. Since this driver takes 1k of space it is necessary to reduce the Temporary Program Area (TPA) of CP/M by 1k with MOVCPM to make room for the additional code.

Although both types of disk drivers will be discussed in this section primary emphasis will be on the floppy driver. No listing is provided for the hard disk driver since the code is very much unique to the type of hard disk drive used. The hard disk drive DCB will be explained in full.

9.1. The Floppy Disk Driver

The standard disk Model 4/4P comes with two 40 track, single-sided disk drives, and can be equipped with two more external drives. One of the primary goals of the BIOS was to support any combination of disk drives and as many CP/M disk formats as possible. In this section we will describe the data structures used for disk I/O as well as the operation of the Floppy Disk driver itself.

Access to disk drives in CP/M is made using the Disk Parameter Header (DPH) and the Disk Parameter Block (DPB). Both of these structures are explained fully in David Cortesi's book *Inside CP/M*, which you got with your copy of CP/M. Our version of CP/M adds a few bytes to the end of the DPB and creates a new data structure called the Disk Device Control Block (DCB).

9.2. DPB Extensions

The standard CP/M DPB is 15 bytes long. Our additions fields follow the standard ones, preserving the original offsets. The added fields are:

**DPBSPT - Offset 15**
This byte contains the number of real sectors per track, NOT the number of 128 byte CP/M sectors per track. It is used mainly by external programs such as DUP who need to know this value for formatting purposes.

**DPBSSZ - Offset 16**
In this byte is stored a code which indicates the true size of a physical disk sector. Only the four IBM standard sizes are supported, using the following values:

- 00 = 128 bytes per sector
- 01 = 256 bytes per sector
- 02 = 512 bytes per sector
- 03 = 1024 bytes per sector

The value in this field is used both externally, mainly for formatting, and internally, in sector deblocking.
DPBDCB - Offset 17
This two-byte address points to the Device Control Block (DCB) of the physical disk drive assigned to this logical disk drive. By using a pointer, instead of putting the actual drive parameters in the DPB, we can assign two or more to the same physical drive. This saves trips to CONFIG when you need to use several different disk formats at the same time.

DPROPT - Offset 19
A collection of bit flags is stored in this byte. These flags are normally set by CONFIG when establishing a format. Bit assignments are as follows:

1. Drive density
   - 0 = Single, 1 = Double
2. Drive sides
   - 0 = Single sided, 1 = Double sided
3. Drive stepping
   - 0 = Normal, 1 = Double step (on 80 track drive)
4. Data status
   - 0 = Normal, 1 = Inverted (Superbrain)
5. Sector numbering on double-sided drive
   - 0 = Same numbers on each side of disk
   - 1 = Side 1 continues where side 0 left off
6. Track numbering on double-sided drive
   - 0 = Tracks same on each side
   - 1 = Even tracks on side 0, odd tracks on side 1
7. Side selection on double-sided drives
   - 0 = Tracks map on alternating sides
   - 1 = Tracks map first on side 0, then on side 1
8. Track usage on side 1 if bit 4 is set to 1
   - 0 = Tracks run from track 0 to innermost track
   - 1 = Tracks run from innermost track back to track 0

It is our hope that these bits provide the means to access any format, no matter how strange. Bits 0 and 1 were initially unused, due to a mistake believed to be in the original BIOS programmer that all double-sided formats alternated from side to side when stepping to a higher track. This was particularly embarrassing since one of our earlier versions of CP/M was among the ones that didn't follow this pattern. As a result the necessary code to handle this condition was put out of the BIOS and had to be added later using a patch program called ADDCP/B.

Most of the bits in the byte are used by the disk driver in the BIOS. Use extreme care in setting them, since the driver does not blindly do it on selected tracks and sectors, but rather makes decisions based on the parameters passed to it and the bits set in this byte.

DPBDID - Offset 20
This last byte of the DPB is used to keep track of what format has been assigned to a drive. A value of zero is used to signal that the drive is not a floppy disk, although the floppy disk driver pays no attention to this field. Values from 1 to 128 refer to entries in the DISK.FDF file. Since that file is subject to change, this byte may become unexpectedly obsolete and point to the wrong format definition. However BIOS space is very tight, and this byte is used by DUP and CONFIG only to extract the format description for the drive, so the danger is small.
9.3. DCB Definitions

There are four Disk DCBs, one for each of four possible physical drives on the Model 4. Conceivably one could add more, but it would serve no practical purpose. Since the DCB is assigned to a logical drive by a pointer in the DPB it is possible for one DCB to serve multiple logical drives.

**DKDDVVR - Offset 0**
This first field in the DCB is actually a 3 byte Z80 JP instruction. The first byte is OC3H, the opcode for JP, followed by the address of the disk driver. We used this technique so that it would be relatively easy to add other drives to the system, such as hard disk drives.

**DKDSEL - Offset 3**
Each of the drives on the Model 4 has a unique select address, indicated by setting one of the four low order bits. Only one bit should be set in this byte, corresponding to the physical address of the drive.

**DKDATT - Offset 4**
In this byte we use bit flags to keep track of the physical attributes of the drive. Bit assignments are:

- 7: 0=Single sided drive
- 1=Double sided drive
- 6: 0=5 1/4 inch drive
- 1=8 inch drive
- 5: Reserved
- 4: Reserved
- 3: Reserved
- 2: Reserved
- 1,0: Drive step rate code
  - 0=6 ms, 1=12 ms, 2=20 ms, 3=30 ms

These bits are normally set by the CONFIG utility.

**DKDSTD - Offset 5**
This byte contains the start-up delay time for the drive in quarter seconds. It is arbitrarily set at 2, which gives 1/2 second of delay. This byte cannot be set by CONFIG, but must be set using DDT.

**DKDSTL - Offset 6**
After a seek operation it is necessary to give the head time to settle before attempting to read or write. The required settle time in milliseconds is stored here, initially set at 15. This byte cannot be changed except by direct patch.

**DKDNTK - Offset 7**
The number of tracks that a drive can physically access is stored here. The driver will not step to any track beyond this limit.
Montezuma Micro CP/M* System Programmer’s Manual

DKDPTO - Offset 8
Since the disk controller in the Model 4/4P has write precompensation controlled by software the driver must know when to turn it on. The value is not arrived at scientifically, but more by divine inspiration. In his experience the author has not had any problems using half the number of tracks plus 2, so that “magic” number is used here.

DKDCTK - Offset 9
With the potential for up to four drives in the system it is necessary for the driver to reset the current track in the disk controller when changing drives. The current track value is stored here, and is normally not written to. One exception is at cold boot time when 0FFH is written in this field for each DCB. A value of 0FFH forces the driver to restore the drive prior to doing any disk I/O.

DKDCSL - Offset 10
In addition to selecting a disk drive the hardware drive select register also sets the density, write precompensation, and side for the drive being accessed. Once all this information has been collected it is stored here for “refreshing” the drive select register.

DKDLTK - Offset 11
Most of the time the track number recorded on the disk will correspond to the track number that the drive read/write head is positioned over. On those occasions where this is not the case, e.g. an 80 track drive used with a 40 track format, this byte holds the logical track that is expected on the disk.

Although this DCB is used only by the BIOS Floppy Disk Driver and the CONFIG utility, it should not be changed. For other disk drivers you may want to create a DCB which meets the requirements of the drive. This was done when adding hard disk drives to CP/M.

9.4. Using the Disk Driver

It is quite possible to use the BIOS Floppy Disk Driver in external programs. The driver is capable of reading or writing a sector, but does not contain code for more exotic functions such as format. The calling setup is as follows:

A Contains function code
  1 = Read sector
  2 = Write sector
BC Contains track number (B should always be 0)
DE Contains sector number (D should always be 0)
  This is the actual sector, i.e. interleave must already be figured before calling the driver.
HL Contains address of data buffer
IX Contains address of DCB for selected drive
IY Contains address of DPB for selected drive

On return the A register contains the status read from the 17xx disk controller status register. All non-error bits are masked off so only error conditions need be checked for.
When using this driver on double-sided disk drives you must be aware that the controller will determine for itself what track, sector, and side to access. This decision will be based on the rules established in field DPBOPT of the DPB pointed to by the IY register. Study this field carefully and set your track and sector numbers accordingly.

9.5. EXBIOS - EXTending the BIOS

As mentioned earlier all possible methods of accessing a disk in CP/M were not handled in the original BIOS. When this fact became known it was not possible to fix the problem with a simple patch due to the fact that the BIOS filled all but 3 bytes of the two tracks originally reserved for the system. The idea of adding another reserved track was rejected, due to the potential problems that it could cause for existing users. Therefore we decided to create a small program that would install the fix into BIOS memory after the system was booted, since it would remain in place until the next full reset.

The instruction where EXBIOS is “hooked” in is shown in the listing of the Floppy Disk Driver, after the label FDBEGN. At the end of the listing the code for the BIOS extension is given. Note that the CONFIG utility knows about EXBIOS and will remove it before saving the configuration to disk. This must be done since the portion of memory that the extension resides in will never be saved in the two reserved tracks. In the next revision this code will be absorbed back into the BIOS where it belongs.

9.6. The Hard Disk Driver

As good as CP/M is, the use of a hard disk drive makes it even better. With disk space in the millions of bytes instead of thousands you can have all of your favorite software available at once. Even better, programs load many times faster and all disk I/O is generally faster.

Like all good things, this convenience carries a price. Since a hard disk drive is not a part of the standard Model 4/4P computer it must be purchased separately. There are dozens of possible configurations, so the code to access the hard disk cannot be economically contained in the BIOS. MOVCPM is used to make a smaller CP/M (no larger than 63k) so that the extra space at the top of memory can be used for the hard disk driver code.

Montezuma Micro offers hard disk drivers for a broad range of hard disk drives, and the code varies according to the type of controller and drive used. For the convenience of external programs we have kept the DPB for hard drives the same as for floppies. The only real difference is in the DPBDID field, which is always set to 0 for a hard disk. Only the memory disk, drive M:, can have a format ID of 0.

The fields of the hard disk drive DCB are unique to that device, although the first field must be the same as that of the floppy disk drive. This is the only link that the BIOS has to the device driver. Here are the fields:

DKDDVR - Offset 0

Like the floppy disk driver, this field is actually a 3 byte JP instruction to the driver routine. The first byte contains 0C3H, and the last two contain the address of the entry point of the hard disk driver.
DKDSEL - Offset 3
This byte contains the drive select bits for the hard disk drive. The actual bit usage will vary from one controller to another, and is of importance only to the driver itself.

DKDCYL - Offset 4
This is a two-byte field containing the 16-bit count of cylinders on the hard disk. CP/M is oblivious to the cylinder concept and works only with tracks. Head positioning on hard disk drives, however, is done by cylinders. Some drivers use this field, others don’t. We recommend that it be kept at this location in the DCB for the convenience of external utilities that may need such information.

DKDOFF - Offset 6
In this two-byte field is kept the 16-bit count of CP/M tracks that precede this logical drive on the physical hard drive. At first glance this might appear to be the same thing as field DPOFF, but that field cannot be used if drive A: is to be located on any track but the very first track of the hard drive. To avoid such restrictions we let CP/M think that each logical hard drive is an entity all to itself, and use the DCB to sort out who lies where.

The above fields are more or less standard for our drivers. Some drivers may have additional fields in the DCB, but these are for driver use only. Any drivers of your own creation should include these standard fields as a minimum. Also your driver MUST use the same calling sequence as the floppy driver, with regard to registers, etc. If it doesn’t the BIOS will not be able to call it correctly.
10. CP/M BOOTS

In addition to providing I/O drivers for CP/M the BIOS performs one other critical function, that of bootstrap loading the operating system. The term “boot” comes from the phrase “pulling yourself up by your own bootstraps.” When the Model 4/4P is first powered up there is no software in RAM. The first level of boot is the ROM, which reads track 0, sector 1, of disk drive 0 into RAM beginning at 4300H. This sector must be in double density. It may be any length, although the 4P will insist on first loading the ROM image for any size other than 256 bytes.

In CP/M there are two boot processes generally referred to as the Cold Boot and the Warm Boot. The Cold Boot is so named because it is activated when the machine is first turned on, i.e. is still “cold.” At other times, usually between programs, a Warm Boot is performed. While the Cold Boot loads the entire CP/M operating system, including the BIOS, the Warm Boot loads only the CCP and the BDOS. The remainder of this section will discuss each of these two routines.

10.1. The Cold Boot

Once the boot sector has been loaded it will proceed to load the CCP, BDOS, and BIOS into their designated locations. Control is then given to the first BIOS vector, which in turn transfers to the label BOOT in the listing. The system stack is established at address 0000H. At first glance this may appear strange, but in fact when the stack is written to it is first decremented. This results in the stack pointer “wrapping around” to the top of memory so that the data is actually written to 0FFFFH downward.

The first order of business is to do a complete reset of all I/O devices, mainly to initialize the associated DCBs, as well as to clear the RAM work areas used by the BIOS. Next drive A: is made the current drive, and the current track number is set invalid on all floppy drives. Drives A:, B:, C:, and D: are set up in the DPH table.

The next routine has been subject to some criticism by CP/M users. It first tests for the presence of the 128k RAM option. If found all 64k of the expansion RAM is set to 0E5H and drive M: is entered into the DPH table. Many users have requested that the RAM drive be formatted (i.e. filled with 0E5H bytes) ONLY if it is found to be corrupt. At present such a test would require a major change in the BIOS and more memory than is currently available. However we will consider this in the next revision.

Finally, if configured for it, the Cold Boot displays the opening “banner” announcing the CP/M size and version. Control then passes to the CCP, which issues the “A>” prompt and begins CP/M operation. If the system was booted from a hard drive (Model 4P with Radio Shack hard disk only) the jump to the CCP is patched to return to the hard drive boot. There the DPH table is modified to reflect the drive configuration, both floppy and hard, and then the jump is made to the CCP.

10.2. The Warm Boot

Like the Cold Boot, the first thing the Warm Boot must do is to establish a stack. However this stack runs from 0080H to 00FFH. It cannot reside in high memory due to the use of the last 128 bytes by the disk I/O routines for internal stack space.
Next it clears the BIOS disk buffer. This step is very important, since different sizes of disk sectors are used. It is possible for a program to terminate with a write operation pending on the sector currently in the disk buffer. By making this call the Warm Boot can be sure that all pending writes have been serviced.

Next a "warm" reset is done. Essentially this just resets the device drivers.

The remainder of the Warm Boot code reloads the CCP and the BDOS from drive A:. Loading begins at track 0, CP/M sector 2, using the assumption that the DPB for A: contains valid information for deblocking. Some CP/Ms access the system tracks in a different manner from the rest of the disk (including version 1.xx of ours), but we have chosen to be consistent throughout the disk. It makes utility writing so much easier.
11. PITFALLS AND TRAPS

After examining what we have done in the BIOS you may be filled with an urge to improve on it. Before you go wading in with a flailing text editor we'd like to tip you off to a few things.

11.1. INTERRUPTS

Interrupts are truly wonderful things. With interrupts one computer can be made to appear to be doing several things at once. Then again they can also cause disasters of truly epic proportions, as well as introduce bugs that are tougher to kill than a New York Cockroach.

We have chosen not to implement interrupts in the BIOS. Why not? Well the first reason is the Model /4P hardware. It dictates that maskable interrupts will generate a ReSTart to location 38H. Coincidentally this is also RST 7 on the 8080, and it is used by some CP/M software, most notably DDT and other debug utilities. While we could release our CP/M with a modified version of DDT it is certain that other programs out there also use RST 7. One of the main goals for our CP/M was to be compatible with the rest of the world, and a conflict over a ReSTart address would make that goal unattainable.

A second reason for not interrupting has to do with memory management. The Model 4/4P has all sorts of memory map possibilities. Interrupting when the wrong map was in could create disaster, so elaborate locking schemes would be necessary to keep them turned off at critical times. This would mean a much larger BIOS, one which was not as robust.

What do we lose by not having interrupts? A keyboard type-ahead buffer is more difficult to do (but not impossible). A steady blinking cursor is real tough, and background I/O such as serial communications is all but impossible. This is only a partial list. There may be other things, too. Systems software is a constant trade-off situation. We are happy with the choices we have made.

11.2. FEATURES UNIQUE TO THE Z80

While thumbing through the listing you Z80 gurus may begin thinking “Hey, I could shorten this code up by using the IX register here, and the alternate registers there.” No doubt you are right, but we have tried to avoid using Z80 features just because they are there.

In the early days of CP/M all programs were written for the 8080 and it was perfectly safe to use Z80 features without fear of overlap. This is no longer true. Newer programs, such as Turbo Pascal, use all of the Z80 features. Using them in the BIOS could lead to surprise crashes on a massive scale. Help stamp out unscheduled Cold Boots and be very conservative in your code.
11.3. RAM USAGE

While perusing CP/M literature you may notice the odd bit of RAM that is reserved for, but not used by, our BIOS. One example of this is locations 0040H through 004FH. It is reserved for the BIOS, but not referenced at all by ours.

Using this area would be unwise, however, because Monte's Window uses the fool out of it. There are other little cracks and crevices in the RAM above and below the BIOS. Enhancement architects are always looking for little crannies to stick their constructions into. Be very, very careful about using these since we sometimes need RAM, too, and we don't know or care what you have used a RAM "hole" for.
12. INDEX

A
Assembler: 1

B
BIOS: 1, 3, 4, 6, 7, 9, 10, 11, 12, 13, 15, 17, 19, 20, 22, 23, 24, 25, 26, 27, 28, 31
Boot: 3, 4, 13, 17, 22, 23, 25, 26, 27
   Cold: 17, 22, 25, 26, 27
   Warm: 3, 13, 25, 26

D
DCB: 4, 9, 10, 11, 12, 13, 15, 19, 20, 21, 22, 23, 24, 25
Device control block: 4, 9, 13, 19, 20
Device driver: 4, 7, 23
   Device driver address table: 4
Device Parameter Block: 19
   DPB: 19, 20, 21, 22, 23, 24, 26
Device Parameter Header: 3
   DPH: 3, 4, 17, 25
Disk I/O: 4, 19, 23, 26

E
EXBIOS: 20, 23

F
Floppy disk driver: 19, 22, 23
Function keys: 9, 10

H
Hard disk driver: 4, 19, 23

I
Initialization: 15
Installation: 4, 7, 19
Interrupts: 27
IOBYTE: 3, 4, 5

K
Keyboard driver: 4, 6, 9, 10
   CAPS lock: 10
   Key definitions: 10

L
Logical device: 5, 6

M
Memory disk: 17, 23

P
Parallel printer driver: 13
Physical device: 5, 6

S
Serial port driver: 15
SPB: 3, 4, 7, 11, 15
System Parameter Block: 3
   BIOS version: 1
   Boot display: 25
   Disk DCB: 4, 21
   DPH table: 4, 17, 25

V
Video Display DCB: 11
   Control codes: 11, 12
   Cursor: 4, 11, 27

Z
Z80 features: 27
Z80 mnemonics: 1
13. THE LISTING

If you're a true System Programmer you've probably already been through the listing of the BIOS before reading the text. You may have noticed that there are some routines missing. For the most part these are mundane little service routines that are not significant to the operation of the BIOS as a whole. Our intent with this manual was not to give you every last byte of source code, but rather a tool with which you could interact with the BIOS.

The bottom line is "This is it!". Please don't write or call us with sad stories about why you need this unpublished routine or that undocumented code. You won't get it. We have a lot of time and money invested in bringing our CP/M this far. Modesty tells us that it could be improved, but good business sense tells us that we should reserve the first option for making those improvements.
TRS-80 Model 4 BIOS Version 2.99+

General Definitions

Copyright (c) (p) 1984
Montezuma Micro
P. O. Box 763009
Dallas, TX 75376-3009

All rights reserved

This BIOS is written for Montezuma Micro CP/M 2.2.

**********************************************************
* CP/M address constants  
**********************************************************

00 D4  BASE    EQU  0D400H ;Base for 64K system
00 D4  CCP    EQU  BASE    ;Base of CCP
06 DC  BDOS   EQU  CCP+806H ;Base of BDOS
00 EA  BIOS   EQU  CCP+1600H ;Base of BIOS
40 00  MSIZE  EQU  (BIOS+1200H)/1024+1 ;Memory size in K bytes

>--- WARNING: BIOS must be 8000H or higher!

00 00  WBJP   EQU  0000H ;BIOS Warm boot vector
03 00  IOBYTE EQU  0003H ;System I/O byte
04 00  CDISK  EQU  0004H ;System current disk drive
80 00  DEBUF  EQU  0080H ;Default disk buffer
2C 00  NRECS  EQU  (BIOS-CCP)/128 ;Number of warm boot recs

**********************************************************
* Model 4 port addresses *
**********************************************************

84 00  MEMCTL EQU  84H ;Memory mapping port
90 00  SOUND  EQU  90H ;Sound control port
E0 00  INTCTL EQU  0E0H ;Interrupt control port
E4 00  NMICTL EQU  0E4H ;Non-maskable interrupt ctrl
E8 00  SERRST EQU  0E8H ;Serial port reset
E9 00  SERRBG EQU  0E9H ;Serial port baud rate gen.
EA 00  SERRUT EQU  0EAH ;Serial port UART ctrl/status
EB 00  SERDAT EQU  0EBH ;Serial port data
EC 00  MISCTL EQU  0ECH ;Miscellaneous function port
F0 00  FDCTLK EQU  0F0H ;Disk command/status
F1 00  FDCTRKE EQU  0F1H ;Disk track
F2 00  FDCESC EQU  0F2H ;Disk sector
F3 00  FDCAI  EQU  0F3H ;Disk data
F4 00  FDCSSL EQU  0F4H ;Disk select
F8 00  PARS DT EQU  0F8H ;Parallel port status/data

**********************************************************
* Model 4 data constants *
**********************************************************

8E 00  KVMIN  EQU  8EH ;Keyboard/Video mapped in
8F 00  KVMOUT EQU  8FH ;Keyboard/Video mapped out

TRS-80 Model 4 BIOS Version 2.99+
Entry vectors & configuration data

EA00

ORG BIOS ;Start BIOS code

**********************************************************
* Standard BIOS jump vectors *
**********************************************************
TRS-80 Model 4 BIOS Version 2.00+ Boot routines

;**********************************************************
;* BIOS Cold Start entry  
;* Input: None.  
;* Output: None - System loaded into RAM  
;**********************************************************
EA4B 31 00 00  ;BOOT
EA4E CD D3 EC  ;CALL
EA51 AF        ;XOR
EA52 32 04 00  ;LD (CDISK),A
EA55 32 3F EB  ;LD (BANNRM),A
EA58 3D        ;DEC
EA59 32 5E F6  ;LD (D0DCB+DKDCTK),A
EA5C 32 6A F6  ;LD (D1DCB+DKDCTK),A
EA5F 32 76 F6  ;LD (D2DCB+DKDCTK),A
EA62 32 82 F6  ;LD (D3DCB+DKDCTK),A
EA65 21 9C F5  ;LD (DLPH),A
EA68 01 10 00  ;LD BC,16
EA6B 22 FD F6  ;LD (DPTHBL),HL
EA6E 09        ;ADD HL,BC
EA72 09        ;ADD HL,BC

;**********************************************************
;* System Parameter Block  
;* This block is used to contain configuration data of  
;* a general nature that is required by the BIOS and  
;* external routines that may need to modify it.  
;**********************************************************

EA33 81  ;SPB EQU $1
EA34 FF  ;SPBIOB DEBF @81H  ;IOBYTE: LPT,TTY,TTY,CRT+0
EA35 02  ;SPBSOM DEBF 2  ;Total # of disk drives +2
EA36 22  ;DEBF 22H  ;BIOS version number +3
EA37 FD F6  ;DEFW DPHTBL  ;DPH table address +4
EA39 55 F6  ;DEFW D0DCB  ;Disk DCB 0 address +6
EA3B 61 F6  ;DEFW D1DCB  ;Disk DCB 1 address +8
EA3D 6D F6  ;DEFW D2DCB  ;Disk DCB 2 address +10
EA3F 79 F6  ;DEFW D3DCB  ;Disk DCB 3 address +12
EA41 73 EC  ;DEFW DDATBL  ;Device Driver Address +14
EA43 9A EE  ;DEFW KBDCB  ;Keyboard DCB +16
EA45 8E 00  ;DEFW VDDCB  ;Video Display DCB +18
EA47 24 F1  ;DEFW PPDCB  ;Parallel Port DCB +20
EA49 72 F1  ;DEFW SPDCB  ;Serial Port DCB +22

TR-S80 Model 4 BIOS Version 2.00+ Boot routines

;**********************************************************
;* System Parameter Block  
;* This block is used to contain configuration data of  
;* a general nature that is required by the BIOS and  
;* external routines that may need to modify it.  
;**********************************************************

Page 34 - ©(p) Copyright 1985 by Montezuma Micro/JBO
LD (DPHTBL+4),HL ; Drive C
LD (DPHTBL+6),HL ; Drive D
LD HL,Ø0000H ; Point to start of RAM
LD A,KVMOUT+60H ; Switch in expansion bank Ø
OUT (MEMCTL),A
LD (HL),3CH ; Plug with inversion of C3H
LD A,KVMOUT ; Switch back to main RAM
OUT (MEMCTL),A
LD A,(HL) ; Get test byte
LD (HL),ØC3H ; Replace in case it changed
CP (HL) ; Is it unchanged?
JR NZ,BOOT1 ; Go if changed - not 128K
LD A,KVMOUT+60H ; Switch in expansion bank Ø
OUT (MEMCTL),A
CALL BOOT2 ; Fill 32K with E5 bytes
ADD HL,HL ; Set HL back to Ø0000H
LD A,KVMOUT+70H ; Switch in expansion bank 1
OUT (MEMCTL),A
CALL BOOT2 ; Fill 32K with E5 bytes
LD A,Ø00H ; Enable drive M message
OUT (MEMCTL),A
LD (BANNRM),A ; Set up DPH for M:
LD HL,DPHM ; Set up lower RAM map
LD A,(SPBSOM) ; Check the signon flag
OR A
CALL NZ,DISPLAY ; Display if requested
LD C,Ø ; Set default drive to A:
JP CCP ; Go to CP/M
LD HL,BANNER ; Point to opening banner
LD A,1AH.07H.16H ; Store an E5 byte
CALL Boot1 ; Restore lower RAM map
LD A,(SPBSOM) ; Check the signon flag
INC HL ; Advance pointer
BIT 7,H ; Check bit 7 of address
RET NZ ; Exit if at 32K
JR Boot2 ; Keep filling
LD HL,H7,E5 ; CP/M signon banner
LD A,1AH,Ø7H,16H
LD A,'TRS-Ø0 Model 4'
LD A,0H
LD A,'k CP/M vers 2.2'
LD A,'(c) (p) 1982 Digital Research Inc.'
LD A,15H,ØDH,ØAH
LD A,'BIOS vers 2.20'
LD A,'BIOS vers 2.20'
BANNER DEFB 1AH,Ø7H,16H
BANNER DEFB 'TRS-Ø0 Model 4'
BANNER DEFB 'k CP/M vers 2.2'
BANNER DEFB '(c) (p) 1982 Digital Research Inc.'
**Page 36 - ©(p) Copyright 1985 by Montezuma Micro/JBO**
TR-80 Model 4 BIOS Version 2.00+ I/O routines for CON: device

;*****************************************************************************
; * Report console status
; * Input: None
; * Output: A=FFH if input present
; * 00H if no input present
;*****************************************************************************
CONIOB    CALL    CONIOB    ;Get CON IOBYTE
          JR      Z,CONST1    ;Go if BAT status
          CALL    IODSP    ;Call I/O dispatcher
          DEFW    TTYSTS    ;TTY status
          DEFW    CRTSTS    ;CRT status
          DEFW    NULSTS    ;BAT status (Dummy entry)
          DEFW    UC1STS    ;UC1 status
          LD      A,(IIOBYTE)    ;Get the IOBYTE
          RRCA
          RRCA
          AND      03H    ;Call I/O dispatcher
          CALL    IODSP    ;TTY input
          CALL    IODSP    ;BAT input (Dummy entry)
          CALL    IODSP    ;UC1 input

;*****************************************************************************
; * Console input
; * Input: None
; * Output: A=Character input from console
;*****************************************************************************
CONIN    CALL    CONIOB    ;Get CON IOBYTE
          JR      Z,READER    ;Go if BAT
          CALL    IODSP    ;Call I/O dispatcher
          DEFW    TTYINP    ;TTY input
          DEFW    CRTINP    ;CRT input
          DEFW    NULINP    ;BAT input (Dummy entry)
          DEFW    UC1INP    ;UC1 input

;*****************************************************************************
; * Console output
; * Input: C=Character to be output to console
; * Output: None
;*****************************************************************************
CONOUT   CALL    CONIOB    ;Get CON IOBYTE
          JR      Z,LIST    ;Go if BAT
          CALL    IODSP    ;Call I/O dispatcher
          DEFW    TTYOUT    ;TTY output
          DEFW    CRTOUT    ;CRT output
          DEFW    NULOUT    ;BAT output (Dummy entry)
          DEFW    UC1OUT    ;UC1 output

(c) Copyright 1985 by Montezuma Micro/JBO - Page 37
TRSI Model 4 BIOS Version 2.00+ I/O routines for LST: device

* Return I/OBYTE value
* Input: None
* Output: A=I/OBYTE value, Z flag set if BAT

EC12 3A Ø3 ØØ  CONIOB  LD  A,(I/OBYTE) ;Get the I/OBYTE
EC15 E6 Ø3  AND  Ø3H ;Isolate CON bits
EC17 FE Ø2  CP  Ø2H ;Check for BAT
EC19 C9  RE!

TRS-80 Model 4 BIOS Version 2.00+ I/O routines for PUN: device

* Return character to PUN device
* Input: C=character to output
* Output: None

EC3A 3A Ø3 ØØ  PUNCH  LD  A,(I/OBYTE) ;Get the I/OBYTE
EC3D Ø7  RLCA ;Isolate PUN bits
EC3E Ø7  RLCA
EC31 E6 Ø3  AND  Ø3H
EC33 CD 64 EC  CALL  IODSP ;Call I/O dispatcher
EC36 79 EC  DEFW  TTYOUT ; TTY output
EC38 81 EC  DEFW  CRTBSY ; CRT busy
EC3A 91 EC  DEFW  LPTBSY ; LPT busy
EC3C 99 EC  DEFW  ULIBSY ; UL1 busy

Page 38 - ©(p) Copyright 1985 by Montezuma Micro/JBO
**TRS-80 Model 4 BIOS Version 2.00+ I/O routines for RDR: device**

* Input from RDR device
* Output: A=character input

```assembly
EC52  3A 03 00
EC55  0F
EC56  0F
EC57  E6 03
EC59  CD 64 EC
EC5C  75 EC
EC5E  9D EC
EC60  A5 EC
EC62  AD EC

**TRS-80 Model 4 BIOS Version 2.00+ General BIOS subroutines**

* I/O dispatch routine
* Input: A=Device code (0-3)
* (SP)=pointer to address table
* Output: None - goes to device routine

```assembly
EC64  E1
EC65  87
EC66  5F
EC67  16 00
EC69  19
EC6A  5E
EC6B  23
EC6C  56
EC6D  EB
EC6E  5E
EC6F  23
EC70  56
EC71  EB
EC72  E9

**Device Driver Address Table**

```assembly
DDATBL EQU $

<table>
<thead>
<tr>
<th>Address</th>
<th>Description</th>
<th>Address</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EC73</td>
<td>TTY definitions</td>
<td>EC75</td>
<td>TTY definitions</td>
</tr>
<tr>
<td>28 F1</td>
<td></td>
<td>30 F1</td>
<td></td>
</tr>
<tr>
<td>EC77</td>
<td></td>
<td>3B F1</td>
<td></td>
</tr>
<tr>
<td>44 F1</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Address</th>
<th>Description</th>
<th>Address</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EC78</td>
<td>CRT definitions</td>
<td>EC7D</td>
<td>CRT definitions</td>
</tr>
<tr>
<td>51 ED</td>
<td></td>
<td>61 ED</td>
<td></td>
</tr>
<tr>
<td>EC7F</td>
<td></td>
<td>46 EF</td>
<td></td>
</tr>
<tr>
<td>EC81</td>
<td></td>
<td>DØ EC</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Address</th>
<th>Description</th>
<th>Address</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EC83</td>
<td>UCl definitions</td>
<td>EC85</td>
<td>UCl definitions</td>
</tr>
<tr>
<td>51 ED</td>
<td></td>
<td>61 ED</td>
<td></td>
</tr>
<tr>
<td>EC87</td>
<td></td>
<td>46 EF</td>
<td></td>
</tr>
<tr>
<td>EC89</td>
<td></td>
<td>DØ EC</td>
<td></td>
</tr>
</tbody>
</table>

**DDSPL** POP HL; Table pointer to HL
ADD A,A; Compute offset
LD E,A; Move offset to DE
LD D,0; Point to address
ADD HL,DE
LD E,(HL); DE=vector pointer
INC HL
LD D,(HL); HL=vector pointer
LD E,(HL); Vector to DE
INC HL
LD D,(HL); HL=driver address
LD D,(HL); HL=driver address
LD (HL); Exit to device driver

73 EC DDATBL EQU $;

<table>
<thead>
<tr>
<th>Address</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EC73</td>
<td>TTY definitions</td>
</tr>
<tr>
<td>28 F1</td>
<td></td>
</tr>
<tr>
<td>EC75</td>
<td>TTY definitions</td>
</tr>
<tr>
<td>30 F1</td>
<td></td>
</tr>
<tr>
<td>EC77</td>
<td></td>
</tr>
<tr>
<td>3B F1</td>
<td></td>
</tr>
<tr>
<td>EC79</td>
<td></td>
</tr>
<tr>
<td>44 F1</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Address</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EC78</td>
<td>CRT definitions</td>
</tr>
<tr>
<td>51 ED</td>
<td></td>
</tr>
<tr>
<td>EC7D</td>
<td>CRT definitions</td>
</tr>
<tr>
<td>61 ED</td>
<td></td>
</tr>
<tr>
<td>EC7F</td>
<td></td>
</tr>
<tr>
<td>46 EF</td>
<td></td>
</tr>
<tr>
<td>EC81</td>
<td></td>
</tr>
<tr>
<td>DØ EC</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Address</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EC83</td>
<td>UCl definitions</td>
</tr>
<tr>
<td>51 ED</td>
<td></td>
</tr>
<tr>
<td>EC85</td>
<td>UCl definitions</td>
</tr>
<tr>
<td>61 ED</td>
<td></td>
</tr>
<tr>
<td>EC87</td>
<td></td>
</tr>
<tr>
<td>46 EF</td>
<td></td>
</tr>
<tr>
<td>EC89</td>
<td></td>
</tr>
<tr>
<td>DØ EC</td>
<td></td>
</tr>
</tbody>
</table>
; LPT definitions
; ****************************
EC8B CB EC  LPTSTS  DEFW  NULSTS ;Null status
EC8D CD EC  LPTINP  DEFW  NULINP ;Null input
EC8F BF F0  LPTOUT  DEFW  PPOUT ;Parallel port output
EC91 B3 F0  LPTBSY  DEFW  PPBSY ;Parallel port busy
; ****************************

; UL1 definitions
; ****************************
EC93 CB EC  UL1STS  DEFW  NULSTS ;Null status
EC95 CD EC  UL1INP  DEFW  NULINP ;Null input
EC97 BF F0  UL1OUT  DEFW  PPOUT ;Parallel port output
EC99 B3 F0  UL1BSY  DEFW  PPBSY ;Parallel port busy
; ****************************

; PTR definitions
; ****************************
EC9B 51 ED  PTRSTS  DEFW  KBSTS ;Keyboard status
EC9D 61 ED  PTRINP  DEFW  KBINP ;Keyboard input
EC9F CC EC  PTROUT  DEFW  NULOUT ;Null output
ECA1 D0 EC  PTRBSY  DEFW  NULBSY ;Null busy
; ****************************

; UR1 definitions
; ****************************
ECA3 28 F1  UR1STS  DEFW  SPSTS ;Serial port status
ECA5 30 F1  UR1INP  DEFW  SPINP ;Serial port input
ECA7 3B F1  UR1OUT  DEFW  SPOUT ;Serial port output
ECA9 44 F1  UR1BSY  DEFW  SPBSY ;Serial port busy
; ****************************

; UR2 definitions
; ****************************
ECAB 28 F1  UR2STS  DEFW  SPSTS ;Serial port status
ECAD 30 F1  UR2INP  DEFW  SPINP ;Serial port input
ECAF 3B F1  UR2OUT  DEFW  SPOUT ;Serial port output
ECB1 44 F1  UR2BSY  DEFW  SPBSY ;Serial port busy
; ****************************

; PTP definitions
; ****************************
ECB3 CB EC  PTPSTS  DEFW  NULSTS ;Null status
ECB5 CD EC  PTPINP  DEFW  NULINP ;Null input
ECB7 46 EF  PTPOUT  DEFW  VDOUT ;Video output
ECB9 D0 EC  PTPBSY  DEFW  NULBSY ;Null busy
; ****************************

; UP1 definitions
; ****************************
ECBB 28 F1  UP1STS  DEFW  SPSTS ;Serial port status
ECBD 30 F1  UP1INP  DEFW  SPINP ;Serial port input
ECBF 3B F1  UP1OUT  DEFW  SPOUT ;Serial port output
ECC1 44 F1  UP1BSY  DEFW  SPBSY ;Serial port busy
; ****************************

; UP2 definitions
; ****************************
ECC3 28 F1  UP2STS  DEFW  SPSTS ;Serial port status
ECC5 30 F1  UP2INP  DEFW  SPINP ;Serial port input
ECC7 3B F1  UP2OUT  DEFW  SPOUT ;Serial port output
ECC9 44 F1  UP2BSY  DEFW  SPBSY ;Serial port busy
; ****************************

; *****************************************
; * Null device drivers                        *
; * Input: None expected                       *
; * Output: None                               *
; *****************************************
ECBB AF  NULSTS  XOR  A ;Null status
ECCC C9   NULOUT  RET ;Null output
ECCD 3E 1A NULINP  LD  A,1AH ;Null input
ECCF C9   RET
ECD0 3E FF NULBSY  LD  A,0FFH ;Null busy
**Device driver for Keyboard**

- **Keyboard device drivers**
- **Input:** None
- **Output:** Dependent on function

---

Return keyboard status in A

```
ED51  3A 9A EE
ED54  B7
ED55  20 07
ED57  CD 6F ED
ED5A  C8
ED5B  32 9A EE
ED5E  F6 FF
ED60  C9

ED61  21 9A EE
ED64  7E
ED65  36 00
ED67  B7
ED68  C0
ED69  CD 6F ED
ED6C  28 FB
ED6E  C9
```

- **Input from keyboard & return key in A**

```
ED6F  F3
ED72  D3 8A
ED74  CD 24 EE
ED7A  11 01 F4
ED7D  06 00
ED7E  21 9D EE
ED82  1A
ED83  4F
ED84  AE
ED85  71
ED86  A1
ED87  20 3D
ED88  04
ED8A  23
ED8B  CB 03
ED8D  F2 82 ED
ED90  3A A5 EE
ED93  5F
ED94  1A
ED95  4F
ED96  2A A6 EE
ED99  7E
ED9A  A1
ED9B  20 0D
ED9D  ED 62
ED9F  22 AA EE
EDA2  21 00 08
EDA5  22 A8 EE
EDA8  18 72
```

---

General keyboard scan - key returned in A if found

```
EDAA  AF
```

©(p) Copyright 1985 by Montezuma Micro/JBO - Page 41
nibiury pointer 10 LIE.
Repeat counter to HL
Increment the count
Save the counter
Get the delay value
Delay long enough?
Exit if no time-out
Clear history for rescan
Save zeroed counter
Set short delay
Go scan again
True scan to C
Do debounce delay
Exit if no key
Save row bit
Save image pointer
Multiply row # by 8
Precomp for shift
Update char position
Shift strobe bit left 1
Loop till it falls off
Point HL at control image
Get table offset
In alpha keys?
Go if not
Control pressed?
Exit if yes
Convert offset to ASCII
Is this '0' key?
Exit if yes
Invert bit 5
Precomp for key
Put in BC
Decode table base to DE
Move to HL, KBDHST to DE
Point to standard table
Table length to BC
Isolate CTRL, SHIFT keys
60 if neither down
Move to SHIFT table
Isolate CTRL key
Go if only SHIFT
Move to CTRL table
Get decoded key in B
Return key to A
Store character in C
Switch out keyboard
; Exit with key
; Calculate offset
; Put in BC
; Decode table base to DE
; Move to HL, KBDHST to DE
; Point to standard table
; Table length to BC
; Isolate CTRL, SHIFT keys
; Go if neither down
; Move to SHIFT table
; Isolate CTRL key
; Go if only SHIFT
; Move to CTRL table
; Get decoded key in B
; Return key to A
; Store character in C
; Switch out keyboard
; Restore the key, if any
; Set Z if no key found
Scan function keys

; Get Function Key Pointer
LD HL, (KBDFKP) ; Is a key active?

; If yes
JR NZ, KBFKC5

; Set DE for rows 0-6
LD DE, 0F47H
; Preset key mask
LD C, E
; Strobe rows 0-6
LD A, (DE)
; Anything down?
OR A
; Go if not
JR Z, KBFKC1
; Must ignore F1, F2, F3, CAPS
LD C, 07H
; Point HL at row 7 image
LD HL, KBDHST + 7
; Set DE for row 7
JR NZ, A70H
; Strobe row 7
LD A, (DE)
; Mask off if necessary
AND C
; Result in C
JR NZ, A70H
; Set changed bits
LD (HL), C
; Save current scan
AND C
; Mask released keys
LD C, A
; Corrected scan to C
CALL KBDBN
; Do debounce delay
RET Z
; Exit if no key down
JR NZ, A70H
; CAPS key down?
LD A, (KBDCFLF, A)
; Go if not
XOR HL, 01H
; Toggle the flag
CALL NZ, VDBEL1
; Save registers
POP BC
; Set for short beep
LD HL, BC
; Beep if locking
LD A, 70H
; Check function keys
AND C
; Exit if none down
RET Z
; Prepare to position
JR NC, KBFKC3
; History pointer to DE
LD DE, HL
; Point HL at Decode table
LD HL, KBDHST + 99
; Set BC to 1 entry length
LD BC, 9
; Back up table pointer
SBC HL, BC
; Check next F key bit
RLCA
; Loop until found
EX DE, HL
; Preload for next round
LD HL, KBDHST + 99
; Get key scan from KDBHST
LD A, (DE)
; Check the SHIFT keys
AND C
; Go if either down
JR NZ, KBFKC4
; Get key scan again
LD A, (DE)
; Check the CTRL key
AND 4
; Go if not pressed
JR Z, KBFKC5
; Move down one group
ADD HL, BC
; Move down one group
ADD HL, BC
; Get next keystroke
LD A, (HL)
; Update pointer
INC HL
; End of definition?
LD (KBDFKP), HL
OR A
; Save def pointer
RET NZ
; Exit if valid key
LD H, A
; Clear the pointer
LD L, A
; Exit with key
LD (KBDFKP), HL

; Debounce a key

; Set time (app. 15ms)
CALL MSDELY
; Do the delay

©(p) Copyright 1985 by Montezuma Micro/JBO - Page 43
; Initialize Keyboard DCB
;
KBINIT CALL CLRMEM ;Clear DCB fields
LD A,(DE) ;Scan keyboard again
AND C ;Mask off released keys
LD HL,0800H ;Reset repeat counter
LD (KBDDLY),HL
;
; Keyboard Device Control Block
;
KBDCB EQU \$ ;Character buffer
KBDBUF DEFB 0
KBDFKP DEFW 0 ;Function Key Pointer
KBDHST DEFB 0,0,0,0,0,0,0,0 ;History for 8 rows
EEP000 00 00 00 00
EENULL 00
KBDPKR DEFB 0 ;Previous Key Row bit
KBDPKI DEFW 0 ;Previous Key Image pointer
KBDDLY DEFW 0800H ;Delay before repeating
KBDRPT DEFW 0 ;Delay between repeats
KBDHST DEFB 1 ;CAPS Lock Flag
KBDCLF DEFB 0
KBDCOD EQU \$ ;Keyboard Decode table
;
Unshifted keys
DEFB '01234567' ;0 1 2 3 4 5 6 7
DEFB '89:;,-./' ;8 9 : , -. /
DEFB 0DH,18H,03H,0BH ;ENTER CLEAR BREAK UP
DEFB 0AH,08H,09H,20H ;DOWN LEFT RIGHT SPACE
;
Shifted keys
DEFB '01"#$%' ;0 1 2 3 4 5 6 7
DEFB '()+=<=>?' ;8 9 : , -. /
DEFB 0DH,7FH,03H,0BH ;ENTER CLEAR BREAK UP
DEFB 0AH,08H,09H,20H ;DOWN LEFT RIGHT SPACE
;
Control keys
DEFB '0>2468' ;0 1 2 3 4 5 6 7
DEFB '[]>:;<=\' ;8 9 : , -. /
DEFB 0DH,7FH,03H,0BH ;ENTER CLEAR BREAK UP
DEFB 0AH,08H,09H,20H ;DOWN LEFT RIGHT SPACE
;
Function Key Definition table (9 bytes per entry)
;
KBFKD DEFB 'F1',0
KBFKD DEFB 'F2',0
KBFKD DEFB 'F3',0
KBFKD DEFB 'F4',0
KBFKD DEFB 'F5',0
KBFKD DEFB 'F6',0
KBFKD DEFB 'F7',0
KBFKD DEFB 'F8',0
KBFKD DEFB 'F9',0
KBFKD DEFB 'SHIFT/F1',0
KBFKD DEFB 'SHIFT/F2',0

TRS-80 Model 4 BIOS Version 2.00+  Device driver for Video Display

**************************************************
* Video Display drivers                           *
* Input:  Dependent on function                   *
* Output: None returned to caller                 *
**************************************************

Output character in C to Video Display
---------------------------------------------------------------------
VDOUT  DI          ;No interrupts
    LD    A,KVMIN  ;Switch Video into RAM
    OUT   (MEMCTL),A
    LD    A,(VDDCHR)  ;Get character at cursor
    CALL  VDPUT   ;Replace it in Video RAM
    CALL  VDPROC  ;Process input character
    CALL  VGDE    ;Get character at cursor
    LD    (VDDCHR),A  ;Save in DCB
    BIT    7,A  ;Is character inverted?
    JR    Z,VDOUT1  ;Go if not
    LD    A,9BH  ;Set in alternate cursor
    VDOUT1 OR  80H  ;Insure reverse video
    CALL  VDPUT   ;Output cursor
    LD    A,KVMOUT  ;Switch out Video
    OUT   (MEMCTL),A
    RET

; Get a character from Video RAM at cursor
---------------------------------------------------------------------
VDGET  CALL  VDCSR  ;Point HL at cursor position
    LD    A,(HL)  ;Get the character
    RET

; Put a character into Video RAM at cursor
---------------------------------------------------------------------
VDPUT  CALL  VDCSR  ;Point HL at cursor position
    LD    (HL),A  ;Output the character
    RET

; Point HL at cursor position in Video RAM
---------------------------------------------------------------------
VDCSR  LD    HL,(VDDROW)  ;Get cursor column & row

; Compute RAM Address for position (L=Row, H=Col)
---------------------------------------------------------------------
VDCRA  PUSH  BC  ;Save work registers
    PUSH  DE
    LD    BC,0F800H  ;Video RAM base to BC
    LD    D,C  ;Row # to DE
    LD    E,L  ;Column # to C
---------------------------------------------------------------------

©(p) Copyright 1985 by Montezuma Micro/JBO - Page 45
EF7F  62
EF80  29
EF81  29
EF82  19
EF83  29
EF84  29
EF85  29
EF86  29
EF87  09
EF88  D1
EF89  C1
EF8A  C9

LD   H,D ;Row # also in HL
ADD  HL,HL ;HL=Row # * 4
ADD  HL,HL
ADD  HL,DE ;HL=Row # * 5 (4+1)
ADD  HL,HL
ADD  HL,HL
ADD  HL,HL
ADD  HL,HL
ADD  HL,BC ;Add video base, Column #
POP  DE ;Restore registers
POP  BC

EF8B  3A 91 F0
VDPROC LD   A,(VDESC) ;Get ESC sequence control
       OR   A ;In ESC sequence?
       JR   NZ,VDESH ;Go if yes
EF8E  79
LD   A,C ;Get the character
EF8F  20 12
JR   C,VDCTL ;Control code?
CP   20H
EF91  79
JR   C,VDESH1 ;Get inverse video mask
EF92  FE 20
CP   'z' ;Must be 'z'
CP   20H
EF94  3A 90 F0
LD   A,(VDDINV) ;Get inverse video mask
EF96  3A 90 F0
OR   C ;Combine with character
EF99  B1
CALL  VDPUT ;Output to Video Display
EF9A  CD 6F EF
EF9D  2A 8E F0
LD   HL,(VDDROW) ;Cursor Column,Row to HL
EFA0  C3 24 F0
JP    VDCRT . ;Cursor right & exit

EF9B  18 22
EF9E  3A 92 F0
VDESH1 LD   A,(VDDDESC) ;Get saved Row
       LD   L,A ;Put it in L
EF9F  79
LD   A,C ;Get input Column
EFBB  D6 20
SUB  20H ;Convert to actual
EFBA  FE 50
CP   80 ;Is column # valid?
EFBC  38 02
JR   C,$+4 ;Skip next if so
EFBE  3E 4F
LD   A,79 ;Move to last column
EFC0  67
LD   H,A ;Put it in H
EFC1  22 8E F0
LD   (VDDROW),HL ;Store as new cursor
EFC4  AF
XOR  A ;Clear state variable
EFC5  18 0E
JR    VDESHX ; and exit
EFC7  79
VDESH2 LD   A,C ;Get the input character
       SUB  20H ;Convert to actual Row
EFCA  FE 18
CP   24 ;Is it valid?
EFCC  38 02
JR   C,$+4 ;Skip next if it is
EFCE  3E 17
LD   A,23 ;Move to last row
EFD0  32 92 F0
LD   (VDESHX),A ;Store in DCB
EFD3  3E 01
LD   A,1 ;Set next state in A
EFD5  32 91 F0
VDESHX LD   (VDDDESC),A ;Save state variable
       RET ; and exit

EFD9  21 93 F0
VDCTL LD   HL,VDCXAT ;HL=Code Address Table
EFDC  06 00
LD   B,0 ;Table offset in BC
EFDE  09
ADD  HL,BC ;Index to routine offset
EFDF  7E
LD   A,(HL) ;Pick up routine offset
EFE0  B7
OR   A ;Is the code defined?

Page 46 - ©(p) Copyright 1985 by Montezuma Micro/JBO
\section*{Sound the built-in speaker}

\begin{verbatim}
; Ignore if not
\end{verbatim}

\begin{verbatim}
; Point HL at base address
\end{verbatim}

\begin{verbatim}
; Add offset for this code
\end{verbatim}

\begin{verbatim}
; Routine address to stack
\end{verbatim}

\begin{verbatim}
; Cursor Column, Row to HL
\end{verbatim}

\begin{verbatim}
; Go to it
\end{verbatim}

\begin{verbatim}
; Set duration counter
\end{verbatim}

\begin{verbatim}
; Set bit 0 on
\end{verbatim}

\begin{verbatim}
; Set pitch counter
\end{verbatim}

\begin{verbatim}
; Crank up a wave
\end{verbatim}

\begin{verbatim}
; Turn bit 0 off
\end{verbatim}

\begin{verbatim}
; Reset pitch counter
\end{verbatim}

\begin{verbatim}
; Let the wave die
\end{verbatim}

\begin{verbatim}
; Count down duration
\end{verbatim}

\begin{verbatim}
; Loop until timeout
\end{verbatim}

\begin{verbatim}
; At top of screen?
\end{verbatim}

\begin{verbatim}
; Go if yes
\end{verbatim}

\begin{verbatim}
; Back up 1 position
\end{verbatim}

\begin{verbatim}
; Exit if not wrap
\end{verbatim}

\begin{verbatim}
; Move to end of line
\end{verbatim}

\begin{verbatim}
; and back up 1 row
\end{verbatim}

\begin{verbatim}
; Move cursor left 1 position
\end{verbatim}

\begin{verbatim}
; Column \\# to A
\end{verbatim}

\begin{verbatim}
; Make it 0 mod 8
\end{verbatim}

\begin{verbatim}
; Move to next tab stop
\end{verbatim}

\begin{verbatim}
; Line overflow?
\end{verbatim}

\begin{verbatim}
; Exit if not
\end{verbatim}

\begin{verbatim}
; Move down 1 line
\end{verbatim}

\begin{verbatim}
; Move cursor down 1 line
\end{verbatim}

\begin{verbatim}
; Increment the row \\#
\end{verbatim}

\begin{verbatim}
; Go if not negative
\end{verbatim}

\begin{verbatim}
; Hold on top line
\end{verbatim}

\begin{verbatim}
; Move cursor up 1 line
\end{verbatim}

\begin{verbatim}
; Back up 1 row
\end{verbatim}

\begin{verbatim}
; Go if not negative
\end{verbatim}

\begin{verbatim}
; Hold on top line
\end{verbatim}

\begin{verbatim}
; Move cursor right 1 position
\end{verbatim}

\begin{verbatim}
; Advance 1 column
\end{verbatim}

\begin{verbatim}
; Get the new column
\end{verbatim}

\begin{verbatim}
; Still on line?
\end{verbatim}

\begin{verbatim}
; Go if yes
\end{verbatim}

\begin{verbatim}
; Move to next line
\end{verbatim}
; Perform Cursor Home
VDHOM LD L,Ø ;Set row # to Ø

; Perform Carriage Return
VDCR LD H,Ø ;Set column # to Ø

; Check cursor position & scroll if necessary
VDCSCK LD A,L ;Get the cursor Row #
CP 24 ;Is it on-screen?
LD (VDDROW),HL ;Save the cursor
RET C ;Exit if on screen
LD L,23 ;Stay on line 23
LD (VDDROW),HL ;Save the cursor
LD HL,ØF800H+Ø ;Point HL at second line
LD DE,ØF800H ;Point DE at top of screen
LD BC,80*23 ;Move 23 lines of video
LD HL,23 ;Set Row=23, Column=Ø
LD HL,0000H ;Set cursor at 0,0
LD (VDDROW),HL ;Save it in DCB

; Erase to end of current line
VDEOL PUSH HL ;Save cursor position
LD H,ØØ ;Set to end of line + 1
CALL VDCRA ;Calculate RAM address
EX DE,HL ;Put in DE
POP HL ;Restore cursor
JR VDEOS1 ;Go clear

; Home the cursor and clear the screen
VDCLS LD HL,0000H ;Set cursor at 0,0
LD (VDDROW),HL ;Save it in DCB

; Erase to end of screen
VDEOS LD VDEOS1 CALL EX ;Save cursor position
LD OR SBC HL,DE JP DE,0FF80H ;Start to DE, end to HL
LD A,(VDDINV) ;Get inverse video mask
OR 20H ;Create a blank
SBC HL,DE ;Compute clear length
LD B,H ;Move length to BC
LD C,L ;Fill memory & exit
JP MFILL

; Turn inverse video OFF
VDIVØ XOR JR VDIV2
AJ ;Clear the flag
GO store it

; Turn inverse video ON
VDIV1 XOR JR VDIV1
AJ ;Clear the flag
GO store it

; Toggle state of inverse video
VDINV LD A,(VDDINV) ;Get inverse video mask
VDINV1 XOR 80H ;Reverse it
VDINV2 LD (VDDINV),A ;Replace in DCB
RET
The document contains code listings and descriptions of programming functions. Here is a plain text representation of the code:

### Video Display Device Control Block

```assembly
; Initialize Video Display DCB fields
VINIT CALL CLRMEM ;Clear DCB fields
DEFW VDDROW, VDCTXAT-VDDDR0
RET

; Video Display Device Control Block

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>8D F0</td>
<td>VDDCB EQU $</td>
</tr>
<tr>
<td>F0BD 00</td>
<td>VDDCHR DEFB</td>
</tr>
<tr>
<td>F0BE 00</td>
<td>VDDROW DEFB</td>
</tr>
<tr>
<td>F0BF 00</td>
<td>VDDCOL DEFB</td>
</tr>
<tr>
<td>F0BG 00</td>
<td>VDDINV DEFB</td>
</tr>
<tr>
<td>F091 00</td>
<td>VDEESX DEFB</td>
</tr>
<tr>
<td>93 F0</td>
<td>VDCXAT EQU $</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>F093 00</td>
<td>DEFB 00H</td>
</tr>
<tr>
<td>F094 00</td>
<td>DEFB 01H</td>
</tr>
<tr>
<td>F095 00</td>
<td>DEFB 02H</td>
</tr>
<tr>
<td>F096 00</td>
<td>DEFB 03H</td>
</tr>
<tr>
<td>F097 00</td>
<td>DEFB 04H</td>
</tr>
<tr>
<td>F098 00</td>
<td>DEFB 05H</td>
</tr>
<tr>
<td>F099 00</td>
<td>DEFB 06H</td>
</tr>
<tr>
<td>F09A 00</td>
<td>DEFB VDBEL-VDCRTL1</td>
</tr>
<tr>
<td>F09B 00</td>
<td>DEFB VDCOLT-VDCRTL1</td>
</tr>
<tr>
<td>F09C 00</td>
<td>DEFB VDTAB-VDCRTL1</td>
</tr>
<tr>
<td>F09D 00</td>
<td>DEFB VDLF-VDCRTL1</td>
</tr>
<tr>
<td>F09E 00</td>
<td>DEFB VDVT-VDCRTL1</td>
</tr>
<tr>
<td>F09F 00</td>
<td>DEFB VDCRT-VDCRTL1</td>
</tr>
<tr>
<td>F0A0 00</td>
<td>DEFB VDCR-VDCRTL1</td>
</tr>
<tr>
<td>F0A1 00</td>
<td>DEFB VDIV-VDCRTL1</td>
</tr>
<tr>
<td>F0A2 00</td>
<td>DEFB VDIV1-VDCRTL1</td>
</tr>
<tr>
<td>F0A3 00</td>
<td>DEFB 00H</td>
</tr>
<tr>
<td>F0A4 00</td>
<td>DEFB 00H</td>
</tr>
<tr>
<td>F0A5 00</td>
<td>DEFB 00H</td>
</tr>
<tr>
<td>F0A6 00</td>
<td>DEFB 00H</td>
</tr>
<tr>
<td>F0A7 00</td>
<td>DEFB 00H</td>
</tr>
<tr>
<td>F0A8 00</td>
<td>DEFB VDEOL-VDCRTL1</td>
</tr>
<tr>
<td>F0A9 00</td>
<td>DEFB VDIV-EVDCRTL1</td>
</tr>
<tr>
<td>F0AA 00</td>
<td>DEFB 00H</td>
</tr>
<tr>
<td>F0AB 00</td>
<td>DEFB 00H</td>
</tr>
<tr>
<td>F0AC 00</td>
<td>DEFB VDEOS-VDCRTL1</td>
</tr>
<tr>
<td>F0AD 00</td>
<td>DEFB VDCLS-VDCRTL1</td>
</tr>
<tr>
<td>F0AE 00</td>
<td>DEFB VDESC-VDCRTL1</td>
</tr>
<tr>
<td>F0AF 00</td>
<td>DEFB 00H</td>
</tr>
<tr>
<td>F0B0 00</td>
<td>DEFB 00H</td>
</tr>
<tr>
<td>F0B1 00</td>
<td>DEFB VDHOM-VDCRTL1</td>
</tr>
<tr>
<td>F0B2 00</td>
<td>DEFB 00H</td>
</tr>
</tbody>
</table>
```

TR-80 Model 4 BIOS Version 2.99+ Parallel Printer Port device driver

---

©(p) Copyright 1985 by Montezuma Micro/JBO - Page 49
PPBSY1  DEC  A
            ;Set A to 00H or FFH

            Output C to parallel port
            PPOUT2  CALL  PPBSY
                    ;Wait for printer ready
                    ;Output a linefeed
                    JR    ZZ.PPOUTX
                    A,(PPDPRL)
                    (PARSDT).A

Print the character in C
PRINT  CALL  PPBSY
JR Z,PRINT
LD A,C
OUT (PARSDT),A

Check for linefeed, count down if so
PPCLF  LD A,0AH
CP  C
JR NZ,PPRLC
;Did we just do one?
;Exit if not
DEC A
LD (PPDLCT),A

Reset line counter if zero, exit
PPRLC  LD A,(PPDLCT)
OR A
JR NZ,PPOUTX
;Exit if not
LD (PPDLCT),A

Save character and exit
PPOUTX  LD A,C
LD (PPDPRL),A

Save character in DCB

Page 50 - ©(p) Copyright 1985 by Montezuma Micro/JBO
PPINIT LD A,(PPDPGL) ;Reset line counter
LD (PPDLCT),A
XOR A ;Kill previous character
LD (PPDPRV),A
RET

; Parallel port DCB
PPDCLB EQU $24F1
PPDPRV DEFB 0 ;Previous character
PPDOPT DEFB 1 ;Option bits
; 0=Suppress LF after CR
; 1=Simulate formfeeds
; 2-7=Reserved

PPDLCDF DEFB $24F1
PPDPGL DEFB 66 ;Page length

; TRS-80 Model 4 BIOS Version 2.00+ Serial Port device driver

;********************************************************************
; * Serial Port device drivers                                        *
; * Input:  Dependent on function                                        *
; * Output:  Dependent on function                                      *
;********************************************************************

; Check for input at Serial Port, return status in A
SPSTS IN A,(SERURT) ;Get UART status
AND $00H ;Isolate data received bit
RET Z ;Exit if nothing
OR $0FFH ;Set status to show input
RET

; Input a byte from the Serial Port
SPINP IN A,(SERURT) ;Get UART status
AND $7FH ;Anything received?
JR Z,SPINP ;Loop if not
IN A,(SERDAT) ;Read data byte
AND $7FH ;Mask off parity bit
RET

; Output a byte to the Serial Port
SPOUT CALL SPBSY ;Is the port busy?
JR Z,SPOUT ;Loop until ready
LD A,C ;Get output byte
OUT (SERDAT),A ;Output it
RET

; Check Serial Port for busy
SPBSY IN A,(SERURT) ;Get UART status
AND $40H ;Ready to Xmit?
RET Z ;Exit if not
LD HL,SPDOPT ;Point to options byte
BIT $00H,(HL) ;Wait for CTS enabled?
JR Z,SPBSY1 ;Go if not
IN A,(SERBST) ;Get secondary status
AND $80H ;Check CTS input bit
XOR $00H ;Invert state of CTS *
; Above changed to NOP in 2.22
RET Z ;Exit if no CTS
SPBSY1 BIT 1,(HL) ;Wait for DSR enabled

©(p) Copyright 1985 by Montezuma Micro/JBO - Page 51
F159 28 07 JR Z,SPBSY2 ;Go if not
F15B DB E8 IN A,(SERRST) ;Get secondary status
F15D E6 40 AND 40H ;Isolate DSR bit
F15F EE 40 XOR 40H ;Invert state of DSR *
; Above changed to NOP in 2.22
F161 C8 RET Z ;Exit if no DSR
F162 F6 FF SPBSY2 OR 0FFH ;Indicate ready state
F164 C9 RET

; Initialize Serial Port

; SPINIT LD A,(SPDBDR) ;Set the baud rate
F165 3A 76 F1 F168 D3 E9 OUT (SERBREG),A
F16A D3 E8 OUT (SERRST),A ;Reset the UART
F16C 3A 77 F1 F16E D3 EA LD A,(SPDCFG) ;Configure primary UART reg
F16F D3 EA OUT (SERURT),A
F171 C9 RET

; Serial Port Device Control Block

; MDADDR ;Set up addresses
F172 72 F1 SPDINT JP SPINIT ;Initialization vector
F175 E0 SPDOPT DEB $ ;Serial Port Options
; Bit 0=Wait for CTS
; Bit 1=Wait for DSR
F176 55 SPDBDR DEB 55H ;Baud rate code
F177 6C SPDCFG DEB 6CH ;UART configuration

; TRS-80 Model 4 BIOS Version 2.09+ I/O routines for drive M:

; MDREAD CALL MDADDR ;Set up addresses
F3B5 CD D7 F3 F3B8 CD ED F3 F3BB ED 5B 25 F7 F3BF 21 DB F9
F3C2 01 80 00 MDWRIT LD HL,(DSBDMA) ;Point HL at record
F3C5 ED B0 MOVREC LD BC,128 ;for 1 record length
F3C7 C9 LDIR ;Move the record
F3C9 8C 128 EX JR ;for 1 record length

; MOVREC LD BC,128 ;for 1 record length
F3C8 2A 25 F7 MDWRIT LD HL,(DSBDMA) ;Point HL at record
F3CB 11 DB F9 LD DE,WKBUF ; & DE at work buffer
F3CE CD C2 F3 CALL MOVREC ;Move record to work buffer
F3D1 CD D7 F3 CALL MDADDR ;Set up addresses
F3D4 EB EX DE,HL ;Switch for write
F3D5 18 16 JR MDMOVE ;Write record & exit

Page 52 - ©(p) Copyright 1985 by Montezuma Micro/JBO
I/O routines for Floppy Drives

**Memory drive address setup routine**
* Input: Information in Select Control Block
* Output: A=Map address select bits
  DE=Internal record buffer address
  HL=Record address in alternate memory map

```
F3D7 21 23 F7
F3DA 7E
F3DB 2B
F3DC 66
F3DD 2E 00
F3DF CB 3C
F3E1 CB 1D
F3E3 F6 06
F3E5 Ø7
F3E6 Ø7
F3E7 Ø7
F3E8 Ø7
F3E9 11 DB F9
F3EC C9
```

**Memory drive data move routine**
* Input: A=Address select bits for move
  HL=Source address for move
  DE=Destination address for move
* Output: 128 bytes moved as requested

```
F3ED F3
F3EE F6 8F
F3FØ D3 84
F3F2 CD C2 F3
F3F5 3E 8F
F3F7 D3 84
F3F9 AF
F3FA C9
```

**Floppy Disk I/O Driver**
* Input: A=Function code
  1 - Read a sector
  2 - Write a sector
  BC=Track number (B should always be 0)
  DE=Sector number (D should always be 0)
  HL=Buffer address
  IX=DCB for selected drive
  IY=DPB for selected drive
* Output: A=Status of operation
  Bits match WD 1791 FDC conventions

```
F3FB F3
F3FC 3D
F3FD 2B Ø7
F3FF 3D
F40Ø 2B 3A
```

Return INOP status for Floppy Disk Drive

```
F402 3E 1Ø
F404 B7
F405 C9
```

©(p) Copyright 1985 by Montezuma Micro/JBO - Page 53
; Read a sector from disk
FDREAD CALL FDBEGN ;Start the I/O operation
RET NZ ;Exit if error
CALL FDRD3 ;Try to read 3 times
RET Z ;Exit if it worked
RET M ;Exit if inoperative
CALL FDJOG ;Jog the head
CALL FDRD3 ;Try 3 more times
RET Z ;Exit if read OK
CALL FDRST ;Restore the drive

; Read a sector with 3 attempts
FDRD3 CALL FDRDSC ;Try to read the sector
RET Z ;Exit if it worked
RET M ;Exit if inoperative
CALL FDRDSC ;Try again
RET Z ;Exit if OK

; Read a sector
FDRDSC PUSH HL ;Save buffer address
LD B,80H ;Set up read command
CALL FDSET ;Start the command
DEFW FDRDS3 ;Termination address
IN A,(FDCCTL) ;Read the status
AND E ;Got a DRQ yet?
JR Z,FDRDS1 ;Loop if not
IN A,D ;Establish byte
LD A,D ;Read first byte
PUSH HL ;Read a sector to disk
LD B,A0H ;Save buffer address
CALL FDSET ;Set up write command
DEFW FDWTS4 ;Start the command
IN A,(FDCSEL) ;Termination address
IN A,D ;Read byte
JR FDRD5 ;Keep reading
POP HL ;Read a byte
AND 9CH ;Exit if OK
RET

; Write a sector to disk
FDWRIT CALL FDBEGN ;Start the I/O operation
RET NZ ;Exit if error
CALL FDWT3 ;Try to write 3 times
RET Z ;Exit if it worked
RET M ;Exit if inoperative
CALL FDIJOE ;Jog the head
CALL FDWT3 ;Try 3 more times
RET Z ;Exit if write OK
CALL FDRST ;Restore the drive

; Write a sector with 3 attempts
FDWT3 CALL FDWTSC ;Try to write the sector
RET Z ;Exit if it worked
RET M ;Exit if inoperative
CALL FDWTSC ;Try again
RET Z ;Exit if OK

; Write a sector
FDWTSC PUSH HL ;Save buffer address
LD B,80H ;Set up write command
CALL FDSET ;Start the command
DEFW FDWTS4 ;Termination address
FDWTS1 IN A,(FDCCTL) ; Read the status
F466 A3 AND E
F467 28 FB JR Z,FDWTS1 ; Got a DRQ yet?
F468 7E OUTI ; Loop if not
F469 ED A3 OUT A,(HL) ; Output first byte
F470 23 INC HL ; Get the next byte in A
F471 ØE F0 LD A,(HL) ; Point C at status reg
F472 ED 58 IN E,(C) ; Loop for second DRQ
F473 E2 6F F4 JP PO,FDWTS2 ; Output the byte
F474 D3 F3 OUT (FDCDAT),A ; Restore C to data port
F475 ØE F3 LD C,FDCDAT ; Establish wait states
F476 7A LD A,D ; Go into wait state
F477 D3 F4 OUTI ; Write a byte
F478 ED A3 ; Keep writing
F479 18 FA JR FDWTS3 ; Restore buffer address
F480 E6 FC POP HL ; Any errors?
F481 C9 RET ; Exit with status

FDWTS2 IN E,(C) ; Select the disk & wait for speed;
F483 FD 7E 13 FDBEGN LD A,(IY+DPBOPT); Get drive option bits
F484 ED 80 AND 80H ; Isolate density
F485 DD B6 Ø3 OR (IX+DKDSEL) ; Combine with select bits
F486 DD 71 ØB LD (IX+DKDLTK),C ; Save logical track #
F487 FD CB 13 76 BIT 6,(IY+DPBOPT) ; Double-sided disk?

>> NOTE ^----- EXBIOS replaces the above instruction with this:

>> NOTE CD 80 FE ØØ CALL BIOSEX ; Call BIOS patch
F492 28 1C JR Z,FDBEG2 ; Go if not
F493 CB 39 SRL C ; Divide track # by 2
F494 FD CB 13 56 BIT 2,(IY+DPBOPT) ; Side 1 same track #?
F495 20 Ø3 JR NZ,FDBEG1 ; Go if not
F496 DD 71 ØB LD (IX+DKDLTK),C ; Save new track #
F497 30 ØF FDBEG1 JR NC,FDBEG2 ; Go if on side Ø
F498 F6 10 OR 10H ; Turn on side 1 select
F499 DD 13 5E BIT 3,(IY+DPBOPT) ; Side 1 sectors biased?
F4A0 28 Ø7 JR Z,FDBEG2 ; Go if not
F4A1 F5 PUSH AF ; Save select bits
F4A2 7B LD A,E ; Get sector #
F4A3 FD 86 ØF ADD A,(IY+DPBSPT) ; Add side 1 bias
F4A4 5F LD E,A ; Restore sector #
F4A5 F1 POP AF ; Restore select bits
F4A6 DD 77 ØA FDBEG2 LD (IX+DKDCSL),A ; Save select bits
F4A7 79 LD A,C ; Get track #
F4A8 FD CB 13 6E BIT 5,(IY+DPBOPT) ; Double stepping drive?
F4A9 28 Ø1 JR Z,FDBEG3 ; Go if not
F4AA 87 ADD A,A ; Compute true track #
F4AB DD BE Ø8 FDBEG3 CP (IX+DKDPTO) ; Precomp needed yet?
F4AC 38 Ø4 JR C,FDBEG4 ; Go if not
F4AD DD CB ØA EE SET 5,(IX+DKDCSL) ; Turn it on
F4AE 57 FDBEG4 LD D,A ; True track # to D
F4AF DB F0 IN A,(FDCCTL) ; Get controller status
F4B0 87 RLCA ; Ready bit to C flag
F4B1 CD 3C F5 CALL FDSEL ; Select the drive
F4B2 3E Ø0 LD A,ØDØH ; Reset the FDC
F4B3 D3 F0 OUT (FDCCTL),A ; Go if drive running
F4B4 FF 30 ØD JR NC,FDBEG6 ; Start-up delay to B
F4B5 D4 46 Ø5 LD B,(IX+DKSTD) ; Delay for 1/4 second
F4B6 3E FA FDBEG5 LD A,250 ; Select again
F4B7 14 ED CALL MSDELY ; Wait for speed
F4B8 CD 3C F5 CALL FDSEL ; Get the sector #
F4B9 10 F6 DJNZ FDBEG5 ; Give to controller
F4BA 7B OUT (FDCSEC),A ; Select

☑️(p) Copyright 1985 by Montezuma Micro/JBO - Page 55
Seek the proper track

Get current track
Give to controller
First access (=FFH)?
Restore the drive if so
Get desired track
Is it legal?
Return INOP if so
Output track to FDC
Save also in DCB
;Re-select the drive
;Get the track #
;Any seek required?
Go if not
Target track $ to A
Is it zero?
Go if yes
Set up seek command
Seek the track
Get logical track #
Give it to controller

; Restore the head for I/O retry
Current track # to D
Force restore
Restore, seek & exit

; Jog the head for I/O retry
Step the head in 1 track
Now step out 1 track

; Perform a step operation
;Save BC
Save step command
Wait 2 ms to be sure
erase turned off
Get drive attributes
Isolate step rate
Combine with command
Restore BC
Issue step command
Reselect the drive
Get the status
Still busy?
Delay & return

; Keep disk selected until not busy
;Select the drive
Get the command
Set delay counter
Count down 16 usec
Loop if not zero
Set up for I/O to FDC

FDSET PUSH HL ;Save buffer address

F54A E5
F54B 3A 66 00
F54C 32 58 F9
F54D 2A 67 00
F54E 22 59 F9
F54F 3E C3
F550 32 66 00
F551 21 88 F5
F552 22 67 00
F553 23
F554 28 02
F555 22 67 00
F556 3E C3
F557 32 66 00
F558 21 88 F5
F559 22 67 00
F55A 3E C3
F55B 32 66 00
F55C 21 88 F5
F55D 22 67 00
F55E 23
F55F 28 02
F560 22 67 00
F561 3E C3
F562 E1
F563 E3
F564 5E
F565 23
F566 56
F567 23
F568 8B
F569 E3
F56A D5
F56B DD 56 0A
F56C CB F2
F56D 1E 02
F56E 0E F3
F56F CD 3C F5
F570 DD CB 04 76
F571 28 02
F572 CB D8
F573 78
F574 CD 42 F5
F575 3E C0
F576 D3 E4
F577 99
F578 C9

FDSET1 LD A,(FDCCTL)

FDNMI EX (SP),HL ;Discard return, save HL

F579 AF
F580 D3 E4
F581 3A 58 F9
F582 32 66 00
F583 2A 59 F9
F584 22 67 00
F585 DB F0
F586 E1
F587 C9

TRS-80 Model 4 BIOS Version 2.00+ Disk tables & parameters

********************************************************************
* Disk Parameter Headers (DPH) for drives A-D & M *
********************************************************************
F59C 85 F6 00 00
F59D 00 00 00
F59E 3B F9 EC F5
F59F D8 F8 40 F7
F5A0 A3 F6 00 00
F5A1 00 00 00
F5A2 5B F9 01 F6
F5A3 F8 F8 A4 F7
F5A4 C1 F6 00 00
F5A5 00 00 00

©(p) Copyright 1985 by Montezuma Micro/JBO - Page 57
Resolved Text:

Drive D parameter header

Drive M parameter header

Offsets used to address Disk Parameter Header fields

Drive Ø parameter block

Drive 1 parameter block

Drive 2 parameter block

Page 58 - ©(p) Copyright 1985 by Montezuma Micro/JBO
; * Disk Device Control Blocks (DCB) for drives 0-3 & M *

; Disk Device Control Blocks (DCB) for drives 0-3 & M

; Offsets used to access Disk DCB fields

00 00  DKDDVR EQU 0  ;Driver address
03 00  DKDSEL EQU 3  ;Drive select bits
04 00  DKDAT EQU 4  ;Drive attribute bits
05 00  DKDSTD EQU 5  ;Drive start-up delay in ms
06 00  DKDSL EQU 6  ;Drive settle time in ms
07 00  DKDNK EQU 7  ;Number of tracks
08 00  DKDPTO EQU 8  ;Precomp turn-on track
09 00  DKDCN EQU 9  ;Current track
0A 00  DKDCS EQU 10  ;Current select bits
Is this 35T SuperBrain?

;Go if not
;Set track count to 35

;At end of side 0?
;Go if yes
;Set Z flag

;Return to BIOS

;Going inside out?

;Double track # &
;force side 1
;Save new track #

;Reset Z flag for return

;Restore drive select
;Restore registers
;Back to BIOS

; >>>---> End of BIOS patch