PCPM11SG.WS4   (= Personal CP/M version 11 System Guide)

- "Personal CP/M Version 1.1 -- System Guide"

(Retyped by Emmanuel ROCHE.)

Foreword

Personal CP/M is a single-user operating system for 8-bit computers that use the Zilog Z-80 microprocessor. Personal CP/M is upward-compatible with its predecessor, CP/M 2.2, and offers more features and higher performance than CP/M Version 2. This manual describes the procedures required to adapt Personal CP/M for a custom hardware environement.

Personal CP/M documentation set

The Personal CP/M documentation set includes the following manuals:

1) "Extension of Sharp MZ-800 Personal CP/M"

2) "Personal CP/M User's Guide"

3) "Personal CP/M Programmer's Guide"

4) "Personal CP/M System Guide"

The "Extension of Sharp MZ-800 Personal CP/M" manual contains explanation of the parts of the User's Guide, Programmer's Guide, and System Guide which change when the Personal CP/M is used with the Sharp MZ-800, and information on the additional utilities available with the Sharp MZ-800.

The "Personal CP/M User's Guide" introduces the Personal CP/M operating system, and tells how to use it.

The "Personal CP/M Programmer's Guide" presents information for application programmers who are creating or adapting programs to run under Personal CP/M.

This manual, the "Personal CP/M System Guide", describes the steps necessary to create or modify a Personal CP/M Basic Input/Output System tailored for a specific hardware environment. This manual assumes that you are familiar with systems programming in Z-80 assembly language, and that you have access to a CP/M 2 system. It also assumes that you understand the target hardware, and that you have functioning disk I/O drivers.

You should be familiar with the "Personal CP/M Programmer's Guide", which describes the system calls used by the application programmer to interface with the operating system. The "Programmer's Utility Guide for the CP/M Family of Operating Systems" documents the assembling, linking, and cross-referencing utilities.

How the "System Guide" is organized

Section 1 of the "Personal CP/M System Guide" is an overview of the component modules of the Personal CP/M operating system.

Section 2 provides a description of system generation for all-RAM and ROM/RAM systems.

Section 3 describes bootstrapping procedures for Personal CP/M.

Section 4 describes the entry points, and the required input and returned parameters of all the modules of the BIOS.

Section 5 describes the disk parameter header and associated tables.

In this manual, boldface characters represent user input.

Table of Contents

(To be done by WS4...)

Tables

(idem)

Figures

(idem)

Section 1: System overview

1.1 Introduction

This section is an overview of the Personal CP/M operating system, with a description of the system components and how they relate to each other. Included is a discussion of memory configurations and supported hardware. The last portion summarizes the creation of a customized version of the Personal CP/M Basic Input/Output System (BIOS).

Personal CP/M provides an environment for program execution on computer systems that use the Zilog Z-80 microprocessor. Personal CP/M provides rapid access to data and programs through a file structure that supports dynamic allocation of space for sequential and random access files.

Personal CP/M supports a maximum of 16 logical floppy or hard disks, or disk-like devices, with a storage capacity of up to 8 megabytes each. The maximum file size supported is 8 megabytes. You can configure the number of directory entries and block size to satisfy various needs.

Personal CP/M is supplied for user memory sizes up to 64 kilobytes. The operating system requires about 6 kilobytes of memory, plus that needed for the BIOS.

1.2 Personal CP/M organization

Personal CP/M is composed of 3 system modules: the Console Command Processor (CCP), the Basic Disk Operating System (BDOS), and the Basic Input/Output System (BIOS). These modules are linked together to form the operating system. They are discussed individually in this section.

1.2.1 Memory layout

The Personal CP/M operating system is designed to reside in the top of available memory. Figure 1-1 illustrates 2 types of memory configurations: ROM/RAM, and an all-RAM system. All or part of the operating system code can reside in ROM, with the remaining portion (data areas) at the top of available RAM. In this event, a gap in memory between RAM and ROM can exist. For systems with all RAM, the entire operating system will be at the top of the available memory (typically 64 kilobytes maximum).

```
                ROM                        All RAM
           +------------+              +------------+
      FFFF | BIOS Code  |         FFFF | BIOS Data  |
           +------------+              +------------+
           | BDOS Code  |              | BDOS Data  |
           +------------+              +------------+
                RAM                    | BIOS Code  |
           +------------+              +------------+
           | BIOS Data  |              | BDOS Code  |
           +------------+              +------------+
           | BDOS Data  |              |    CCP     |
           +------------+              +------------+
           |    CCP     |              |            |
           +------------+              |            |
           |            |              |            |
           |            |              |            |
      0100 |    TPA     |         0100 |    TPA     |
           +------------+              +------------+
      0000 | Page Zero  |         0000 | Page Zero  |
           +------------+              +------------+
```

Figure 1-1. Typical Personal CP/M memory layout

1.2.2 Console Command Processor

The Console Command Processor (CCP) provides the user interface to Personal CP/M. The CCP uses the BDOS to read user commands and load programs, and provides several built-in user commands. It also provides parsing of command lines entered at the console. Typically, the standard CCP autoloads the Visual CCP (VCCP).

1.2.3 Basic Disk Operating System

The Basic Disk Operating System (BDOS) provides operating system services to applications programs and to the CCP. These include character I/O, disk file I/O (the BDOS disk I/O operations comprise the Personal CP/M file system), and others.

1.2.4 Basic Input/Output System

The Basic Input/Output System (BIOS) is the interface between Personal CP/M and its hardware environment. All physical input and output is done by the BIOS. It includes all physical device drivers, tables defining disk characteristics, and other hardware-specific functions and tables. The CCP and BDOS do not change for different hardware environment, because all hardware dependencies have been concentrated in the BIOS. Each hardware configuration needs its own BIOS. Section 4 describes the BIOS functions in detail. Section 5 discusses the disk parameter and associated tables, and blocking/deblocking algorithms.

1.3 Input/Output devices

Personal CP/M recognizes 2 basic types of I/O devices: character devices, and disk drives. Character devices are serial devices that handle one character at a time. Disk devices handle data in units of 128 bytes, called logical sectors, and provide a large number of physical sectors which can be accessed in random, non-sequential order. Logical and physical sector sizes can be different. In fact, real systems might have devices with characteristics different from disks, such as a block-accessible, random-access tape cassette device. It is the BIOS's responsibility to resolve differences between the logical device models and the actual physical devices.

1.3.1 Character devices

Character devices are Input/Output devices that accept or supply streams of ASCII characters to the computer. Typical character devices are consoles, printers, and modems. In Personal CP/M, operations on character devices are done one character at a time.

1.3.2 Disk drives

Disk drives are used for file storage. They are organized into sectors and tracks. Each logical sector contains 128 bytes of data. (If physical sector sizes other than 128 bytes are used on the actual disk, then the BIOS must do a logical-to-physical mapping to simulate 128-byte logical sectors to the rest of the system.) All disk I/O in Personal CP/M is done on one-sector units. Usually, a track or cylinder of a disk is a group of physical sectors. The number of sectors on a track is a constant depending on the particular device. (The characteristics of a disk device are specified in the Disk Parameter Block for that device. See Section 5 for more information.)

To locate a particular physical sector, the disk, track number, and sector number must all be specified.

1.4 System generation and cold start operation

Generating a Personal CP/M system is done by linking together the CCP, BDOS, and BIOS to create the operating system.

Section 2 discusses how to create the operating system.

The bootstrap process is discussed in Section 3.

Section 2: System generation

2.1 Overview

This section describes how to build a custom version of Personal CP/M by combining your BIOS with the BDOS supplied by Digital Research. Section 3 describes how to boot the system.

This section assumes that you have access to a working 8-bit CP/M system capable of reading the standard single-sided, single-density 8-inch disk on which Personal CP/M is distributed. You should also be able to create the media (disks, disk-like devices, or ROMs) that the target system will use. It is also assumed that the BIOS is written with an assembler that generates a REL format relocatable object file compatible with the Digital Research LINK-80 linker.

The Personal CP/M operating system is generated by using the linker to resolve external label references between the BDOS and BIOS, and to bind them and the CCP to absolute memory locations.

2.2 Creating a Personal CP/M system file

The CCP and the BDOS for Personal CP/M are distributed on the following 3 files:

1. CCP.REL -- for use with all systems

2. BDOSH.REL -- for use with systems in which the BDOS and BIOS are loaded into and executed from RAM

3. BDOSL.REL -- for use with systems in which the BDOS and BIOS are executed in ROM

You must link your BIOS with one of the two BDOS files. The BDOSH.REL file is used in systems in which the data segment is linked to a higher address than the code segment, as is typical of systems that execute out of RAM. The BDOSL.REL file is used in systems in which the data segment, which must reside in RAM, is linked to a lower address than the code segment, as is the case in a system where BDOS and BIOS execute out of ROM at the top end of the address space.

Each of the Personal CP/M elements, CCP, BDOS, and BIOS, must begin on a page boundary; that is to say: at an address that is a multiple of 100H. The BDOS contains linkage information that automatically forces the BIOS to begin on a page boundary.

For systems in which the BDOS and BIOS execute out of ROM, the BIOS data segment must consist of 'define storage' pseudo-ops only. Any data that must be initialized at cold or warm boot time should be transferred from read-only images of the data in the BIOS code segment.

2.2.1 All-RAM systems

To generate a Personal CP/M operating system image file that can be loaded into RAM at, or near, the top of the memory address space, the folllowing procedure should be used:

1. Determine the highest page boundary on which the BDOS can be located. This is done by adding the size of the BDOS code segment (1100H) and the BDOS data segment (00BFH for BDOSH.REL), plus the size of your BIOS code and data segments. For example, if your BIOS code segment is 0A23H bytes, and the data segment is 0280H bytes, then the following memory map represents the logical arrangement of the Personal CP/M system within memory:

```
                FFFF +-----------+
                     | BIOS Data |
                FCE2 +-----------+
                     | BDOS Data |
                FC23 +-----------+
                     | BIOS Code |
                F200 +-----------+
                     | BDOS Code |
                E100 +-----------+
```

Figure 2-1. All-RAM system configuration

2. Link the BDOS and BIOS together with the following command:

   A>**link pcpm[le100]=bdosh,bios**

   See the "Programmer's Utilities Guide" for LINK-80 command line options. This creates the file PCPM.COM, which contains an absolute image of the object code to be loaded at 0E100H, rather than the standard COM file, which contains an image of the object code to be loaded at 0100H. Note that the BDOS data segment is not required to start on a page boundary in this case.

3. Link the CCP to reside at 0800H less than the load address used in the previous LINK command:

   A>**link ccp[ld900]**

4. The CCP.COM and PCPM.COM files, together with a Cold Boot Loader, can now be written to the system area of the storage media for the target computer. A typical computer system executes a small loader program from ROM, that loads the Cold Boot Loader in from the system area of the storage medium. The Cold Boot Loader then loads the CCP and BDOS/BIOS to the addresses that they are linked to, and finally transfers control to the cold boot entry point of the BIOS.

2.2.2 ROM/RAM systems

To generate a Personal CP/M operating system image file that can execute from ROM at, or near, the top of the memory address space, the following procedure should be used:

1. Determine a page boundary in ROM at which to locate BDOS. Do this by adding the size of the BDOS code segment (1100H) and the size of your BIOS code segment. From the size of the BDOS data segment (00CCH for BDOSL.REL) plus the size of your BIOS data segment, determine a page boundary near the top of RAM at which to locate the data segments. Assuming an 8 kilobytes ROM at the top of the address space, and a BIOS with the same size segments as in the "All-RAM" example, then the following memory map represents the logical arrangement of the Personal CP/M system within memory:

```
            FFFF +-----------+--+
                 | Unused ROM|  |
                 +-----------+  |
                 | BIOS Code |  +--> ROM
            F100 +-----------+  |
                 | BDOS Code |  |
            E000 +-----------+--+
                 | BIOS Data |  |
            DCCC +-----------+  +--> RAM
                 | BDOS Data |  |
            DC00 +-----------+--+
```

Figure 2-2. ROM/RAM system configuration

2. Link the BDOS and BIOS together with the command:

   A>**link pcpm[ldc00,ddc00,pe000]=bdosl,bios**

   This creates the file PCPM.COM, which contains an absolute image of the object data and code to be loaded at 0DC00H, rather than the standard COM file, which contains an image of the object code to be loaded at 0100H. Note that the BDOS data segment is required to start on a page boundary in this case.

3. Link the CCP to reside at 0800H less than the load address used in the previous LINK command:

   A>**link ccp[ld400]**

4. The first part of the PCPM.COM file contains an image of the data segments of BDOS and BIOS. The first 2*n sectors of the file, where n

is the number of pages difference between the data address and the program address in the LINK command for PCPM.COM, must be discarded by your utility program that creates the ROM. This is because the data segments reside in RAM, and must be treated as uninitialized (see Section 2.2). In the example above, 8 sectors (2*4 pages) would be discarded. The reaminder of the PCPM.COM file is then programmed into the ROM.

5. The CCP.COM file, which needs to be reloaded at every Warm Boot, can now be written to the system area of the storage media for the target computer. Another possibility, provided that there is sufficient room (0800H bytes) left over in the ROM, is to store a copy of the CCP in ROM and move it to its execution address at Cold and Warm Boot times.

Section 3: Bootstrap procedures

The bootstrap process involves the following 3 procedures:

1) Do any necessary preliminary hardware initialization.

2) Get the executable object code of the Personal CP/M operating system into memory for execution.

3) Transfer control to the BOOT entry point of the BIOS.

If Personal CP/M is executing out of RAM, the cold boot loader must load the CCP, BDOS, and BIOS into memory at the addresses to which they were linked from the system area of the computer's disk, or disk-like storage media.

If Personal CP/M is executing out of ROM, the BIOS has the responsibility of loading the CCP into memory at cold and warm boot. As mentioned in Section 2, the BIOS is also responsible for initializing any RAM data areas necessary to its operation.

Section 4: BIOS functions

4.1 Introduction

All Personal CP/M hardware dependencies are concentrated in subroutines that are collectively referred to as the Basic Input/Output System (BIOS). A Personal CP/M system implementor can tailor Personal CP/M to fit nearly any Zilog Z-80 operating environment. This section describes the calling conventions and parameters of each BIOS function, and the actions that it must perform.

4.2 BIOS entry points

Entry to the BIOS is through a jump table located at the beginning of BIOS and labels declared PUBLIC. For Personal CP/M, there are 17 fixed jump vectors, with additional functions being defined as PUBLIC. The 17 jump vectors are listed in Table 4-1, and the PUBLIC routines are listed in Table 4-2. The BIOS subroutines can be empty for certain functions (contain a single RET instruction) during reconfiguration of Personal CP/M, but the entries must be present in the jump vector and PUBLIC declarations as well.

Table 4-1. Standard BIOS functions

Function	Input	Output
BOOT	None	None
WBOOT	None	None

CONST	None	A=0FFH if ready
		A=00H if not ready

CONIN	None	A=Character
CONOUT	C=Character	None
LIST	C=Character	None
AUXOUT	C=Character	None
AUXIN	None	A=Character
HOME	None	None

SETDSK	C=Drive (0 through 15)
	E=initial specify flag
		HL=DPH Address
		HL=00H if invalid drive

SETTRK	BC=Track Number	None
SETSEC	BC=Sector Number	None
SETDMA	BC=DMA Address	None

READ	None	A=00H if no Error
		A=01H if Non-recoverable Error

WRITE	C=Deblocking code	A=00H if no Error
		A=01H if Non-recoverable Error

LISTST	None	A=00H if not ready
		A=0FFH if ready

SECTRN	BC=Logical Sector Number	HL=Physical Sector Number
		DE=Translation Table Address

Table 4-2. PUBLIC BIOS subroutines

Function	Input	Output

?AUXIS	None	A=00H if not ready
		A=0FFH if ready

?AUXOS	None	A=00H if not ready
		A=0FFH if ready

?FLUSH	None	A=00H if no error
		A=01H if physical error
		A=02H if disk R-O

?DISCD	None	None

?MOV	HL=Destination Address
	DE=Source Address
		HL & DE point to next bytes
		following MOVE.

?DSCRF	DE=SFB Address	None

?BYTBC *	DE=COPY Block Address	A=00H implemented copy
		A=0FFH not implemented

?BYTBA *	DE=ALTER Block Address	A=00H successful alter
		A=0FFH not implemented

(* = Not supported by the Sharp MZ-800 Personal CP/M.)

All simple character I/O operations are assumed to be performed in ASCII, both uppercase and lowercase. With some programs, an end-of-file condition for an input device is given by an ASCII Ctrl-Z (1AH). Peripheral devices are seen by Personal CP/M as logical devices, and are assigned physical devices within the BIOS.

To operate, the BDOS needs the CONST, CONIN, CONOUT, ?FLUSH, and ?MOV subroutines (the LIST, AUXIN, and AUXOUT subroutines may be used by PIP, but not by the BDOS). The initial version of BIOS may have empty subroutines for the remaining ASCII devices.

The characteristics of each device are as follows:

CONSOLE (CON:)
The principal interactive console, that communicates with the user. Typically, the CONSOLE is a memory-mapped video display.

LIST (LST:)
The principal listing device, if it exists in your system. This is an output-only function.

AUXILIARY INPUT (AUXIN:)
An auxiliary input device, such as serial I/O, paper tape reader, modem, or tape storage peripheral. This is an input-only function.

AUXILIARY OUTPUT (AUXOUT:)
An auxiliary output device, such as serial I/O, paper tape punch, modem, or tape storage peripheral. This is an output-only function.

A single peripheral can be simultaneously the LST:, AUXIN:, or AUXOUT: device. If no peripheral device is assigned as the LST:, AUXIN:, or AUXOUT: device, the BIOS that you create should give an appropriate error message. This prevents the system from hanging if the device is accessed by PIP or some user program.

When the BDOS calls a BIOS function, certain registers will contain information (entry parameters), and are described in the following paragraphs. Also, specific registers are used to return information to the BDOS (returned values). The BIOS returns single-byte results in register A, and double-byte values in register pair HL. For reasons of compatibility, register A = L, and register B = H upon return, in all cases. The size of the result depends on the particular function.

4.3 BDOS entry points

The BDOS contains 3 PUBLIC entry points: ?bdosc, ?bdosw, and ?bdos. The ?bdosc entry point is called by the BIOS Cold Boot code (see the description of the

BOOT entry point). The ?bdosw entry point is called by the BIOS Warm Boot code (see the description of the WBOOT entry point). Finally, the ?bdos entry point is used as the address of the jump instruction written to location 0005H at both Cold and Warm Boot time.

4.4 BIOS entry descriptions

```
        BIOS Function: BOOT
        Get control from Cold Boot Loader, and initializes system.
        Entry Parameters: None
        Returned  Values: None
```

The BOOT entry point gets control from the Cold Start Loader or Power-On/Reset code, and is responsible for the following actions:

1. Do any remaining system hardware initialization.
2. Load the CCP, if it was not loaded by the Cold Start Loader.
3. Display a sign-on message (optional).
4. Set 0000H to jump to BIOS WBOOT entry point.
5. Set 0003H to 00H to default to the standard 'A>' CCP, or to 01H to default to the Visual CCP.
6. Set 0005H to jump to ?bdos.
7. Call the ?bdosc entry point in BDOS.
8. Load register C with the default user number in the high nibble, and the default drive number in the low nibble.
9. Jump to CCP+0003H for the standard CCP, or to CCP+0000H for the Visual CCP.

(ROCHE> Note that no mention is made of a STARTUP.SUB file... Since experience has demonstrated that STARTUP.SUB files are useful, it is recommended that you implement them. One way would be to use the fact that, when the CCP start executing, it checks for any $$$.SUB file present on the disk in drive A. If true, it executes it. So, one could add a piece of code (see the COPY.ASM sample) copying STARTUP.SUB in file $$$.SUB. One drawback: since XSUB is not present, this trick can not pass any option to a program. But SUB files can be nested, so STARTUP.SUB could then contain "SUBMIT COLDBOOT.SUB".)

Table 4-3 gives a description of the locations in Page Zero (0000H through 00FFH) that are used by BOOT and other portions of Personal CP/M.

Table 4-3. Memory Page Zero definitions

Format: Locations
        Contents

0000-0002H
Contains a jump instruction to the warm start entry point. This permits a programmed restart (JMP 0000H) that was commonly used under CP/M 2.2.

0003H
Used as the "VCCP Flag": if clear (00H), then jump to standard CCP; if set (01H), then load and execute the Visual CCP.

0004H
Current default user number (high nibble), and current default drive number (low nibble).

0005-0007H
Contains a jump instruction to the BDOS. A CALL 00005H provides the primary entry point to the BDOS described in the "Personal CP/M Programmer's Guide".

0008-0027H
Interrupt locations 1 through 5 are not used.

0030-0037H
Interrupt location 6, not currently used, but reserved in case your microcomputer uses Z-80 Mode 1 interrupts.

0038-003AH
Interrupt location 7; contains a jump instruction into the SID program when you debug a program, but is not otherwise used by Personal CP/M.

003B-003FH
Not currently used; reserved.

0040-004FH
A 16-byte area reserved for scratch by BIOS, but is not used for any purpose in the distribution version of Personal CP/M.

0050-005BH

Not currently used; reserved.

005C-007CH
Default file control block produced for a transient program by the CCP.

007D-007FH
Optional default random record position.

0080-00FFH
Default 128-byte disk buffer. Also filled with the command line when a transient is loaded under the CCP.

```
        BIOS Function: WBOOT
        Get control when a warm start occurs.
        Entry Parameters: None
        Returned  Values: None
```

The WBOOT entry point gets control whenever a Warm Boot occurs. That is to say: a user program jumps to 0000H or calls BDOS with register C set equal to zero, and is responsible for the following actions:

1. Load the CCP.
2. Set 0000H to jump to BIOS WBOOT entry point.
3. Set 0005H to jump to ?bdos.
4. Call the ?bdosw entry point in BDOS.
5. Load register C with the contents of 0004H.
6. If 0003H equals 00H, then jump to CCP+0003H; otherwise, jump to CCP+0000H.

```
        BIOS Function: CONST
        Sample the status of the console input device.
        Entry Parameters: None
        Returned  Values: A=0FFH if a con char is ready to be read
                          A=00H if no con char is ready to be read
```

Read the status of the currently assigned console device, and return 0FFH in register A if a character is ready to be read, or 00H if a character is not ready.

```
        BIOS Function: CONIN
        Read a character from the console.
        Entry Parameters: None
        Returned  Values: A=console character
```

Read the next console character into register A, with no parity. If no console character is ready, wait until a character is available before returning.

```
        BIOS Function: CONOUT
        Output a character to console.
        Entry Parameters: C=character
        Returned  Values: None
```

This function sends the character from register C to the console output device. The character is in ASCII. You might need to include a delay, or filler characters, for a Line Feed or Carriage Return if your console device requires some time interval at the end of the line.

```
        BIOS Function: LIST
        Output character to list device.
        Entry Parameters: C=character
        Returned  Values: None
```

This function sends an ASCII character from register C to the currently assigned listing device. If your list device requires some communication protocol, it must be handled here.

```
        BIOS Function: AUXOUT
        Output a character to the auxiliary output device.
        Entry Parameters: C=character
        Returned  Values: None
```

This function sends an 8-bit character from register C to the currently assigned auxiliary output device.

```
        BIOS Function: AUXIN
```

```
         Read a character from the auxiliary input device.
         Entry Parameters: None
         Returned  Values: A=character
```

This function reads the next 8-bit character from the currently-assigned auxiliary input device into register A.

```
         BIOS Function: HOME
         Select track zero of the specified drive.
         Entry Parameters: None
         Returned  Values: None
```

This function positions the disk head of the currently specified disk to the track zero position. Usually, you can translate the HOME call into a call on SETTRK with a parameter of zero.

```
         BIOS Function: SETDSK
         Set specified disk drive.
         Entry Parameters: C=disk drive (0 through 15)
                           E=initial specify flag
         Returned  Values: HL=address of the DPH if drive exists
                           HL=0000H if drive does not exist
```

Register C contains the disk drive number for further operations. Register C contains 0 for drive A, 1 for drive B, and so on up to 15 for drive P.

On each disk spec, SETDSK must return in HL the base address of a 16-byte area called the Disk Parameter Header (DPH), as described in Section 5. For standard floppy disk drives, the contents of the header and associated tables do not change. The program segment included in the sample BIOS performs this operation automatically.

If there is an attempt to specify a non-existent drive, SETDSK returns HL=0000H as an error indicator. Although the function must return the header address on each call, it may be advisable to postpone the physical disk specify operation until an I/O function (seek, read, or write) is actually performed. Disk specify operations can occur without performing any disk I/O, and many controllers will unload the head of the current drive before specifying the new drive. This could waste time, and cause an excessive amount of noise and head wear. The least-significant bit of register E is zero if this is the first occurence of the drive specify since the last cold or warm start.

```
         BIOS Function: SETTRK
         Set specified track number.
         Entry Parameters: BC=track number
         Returned  Values: None
```

Register pair BC contains the track number for a subsequent disk access on the currently selected drive. The sector number in BC is the same as the number returned from the SECTRN entry point. You can choose to seek the selected track at this time, or delay the seek until the next READ or WRITE operation actually occurs. Register BC can take on values in the range 0-76, corresponding to valid track numbers for standard 8-inch floppy disk drives, and 0-65535 for non-standard disk subsystems.

```
         BIOS Function: SETSEC
         Set specified sector number.
         Entry Parameters: BC=sector number
         Returned  Values: None
```

Register pair BC contains the sector number for the subsequent disk access on the currently selected drive. This number is the value returned by SECTRN. Usually, actual sector selection is delayed until a READ or WRITE operation occurs. This number remains in effect until another SETSEC function is performed.

```
         BIOS Function: SETDMA
         Set address for subsequent disk I/O.
         Entry Parameters: BC=Direct Memory Access address
         Returned  Values: None
```

Register pair BC contains the Direct Memory Access (DMA) address for the subsequent READ or WRITE operation. For example, if BC=0080H when BDOS calls SETDMA, then the subsequent WRITE operation gets its data from 0080H through 00FFH (the default 128-byte disk buffer), until the next call to SETDMA occurs. The initial DMA address is assumed to be 0080H. The controller need not actually support Direct Memory Access. If, for example, all data transfers are through I/O ports, the BIOS that is constructed uses the 128-byte are starting at the selected DMA address for the memory buffer during the subsequent READ or WRITE operations.

```
        BIOS Function: READ
        Read a sector from the specified drive.
        Entry Parameters: None
        Returned  Values: A=00H if no errors occurred
                          A=01H if non-recoverable error condition encountered
```

Assuming that the drive has been specified, that the track and sector have been set, and that the DMA address has been specified, the READ subroutine attempt to read one sector. The following error codes will be returned in register A:

00H = no errors detected
non-zero = non-recoverable error condition detected

Personal CP/M responds only to a zero or non-zero value. If an error occurs, BIOS should attempt at least 10 retries, to see if the error is recoverable. When an error is reported, the BDOS will output the message "BDOS ERR ON x: BAD SECTOR". The operator then has the option of typing a RETURN to ignore the error, or Ctrl-C to abort.

```
        BIOS Function: WRITE
        Write a sector to the specified drive.
        Entry Parameters: C=00H if normal sector write
                          C=01H if write to directory sector
                          C=02H if write to the first sector of
                                a new data block.
        Returned  Values: A=00H if no error occurred
                          A=01H if non-recoverable error occurred
```

Write the data from the currently selected DMA address to the currently specified drive, track, and sector. Upon each call to WRITE, the BDOS provides the same error codes as the READ function.

As in READ, the BIOS should attempt several retries before reporting an error.

```
        BIOS Function: LISTST
        Return the ready status of the list device.
        Entry Parameters: None
        Returned  Values: A=00H if list device is not ready
                          A=0FFH if list device is ready
```

The BIOS LISTST function returns the ready status of the list device.

```
        BIOS Function: SECTRN
        Translate sector number given translate table.
        Entry Parameters: BC=logical sector number
                          DE=translate table address
        Returned  Values: HL=physical sector number
```

This function performs logical-to-physical sector translation to improve the overall response time of Personal CP/M. Standard Personal CP/M is shipped on a single-sided, single-density 8-inch disk with a "skew factor" of 6, where 6 physical sectors are skipped between each logical read operation. This skew factor allows enough time between sectors for most programs to load their buffers without missing the next sector. In particular computer systems that use fast processors, memory, and disk subsystems, the skew factor can be changed to improve overall response time. However, you should maintain a single-density IBM-compatible version of Personal CP/M for information transfer into and out of the computer system, using a skew factor of 6.

In general, SECTRN receives a logical sector number relative to zero in register BC, and a translate table address in register pair DE. The sector number is used as an index into the translate table. Register pair HL returns the resulting physical sector number. For standard systems, the table and indexing code are provided in the sample BIOS, and need not be changed.

For the rest of this section, the BIOS entry points are defined as PUBLICs.

```
        BIOS Function: ?AUXIS
        Return input status of auxiliary port.
        Entry Parameters: None
        Returned  Values: A=0FFH if ready
                          A=00H if not ready
```

The ?AUXIS routine checks the input status of the auxiliary port. This entry point allows full polled handshaking for communications support using an auxiliary port.

```
        BIOS Function: ?AUXOS
        Return the output status of auxiliary port.
        Entry Parameters: None
        Returned  Values: A=0FFH if ready
                          A=00H if not ready
```

The ?AUXOS routine checks the output status of the auxiliary port. This entry point allows full polled handshaking for communications support using an auxiliary port.

```
        BIOS Function: ?FLUSH
        Force physical buffer flushing for user-supported deblocking.
        Entry Parameters: None
        Returned  Values: A=00H if no error occurred
                          A=01H if physical error occurred
                          A=02H if disk is Read-Only
```

The ?FLUSH buffer entry point allows the system to force physical sector buffer flushing when your BIOS is performing its own record blocking and deblocking.

The BDOS calls the ?FLUSH routine to ensure that no dirty buffers remain in memory. The BIOS should immediately write any buffers that contain unwritten data.

Note: If you do not implement ?FLUSH, the routine must return a zero in register A. This can be accomplished by:

```
        XRA     A
        RET
```

```
        BIOS Function: ?DISCD
        Discard deblocking buffers.
        Entry Parameters: E=drive (0=A, 1=B, ..., 15=P)
        Returned  Values: None
```

This function must discard the contents of the deblocking buffers for the specified drive, or set a flag indicating that the buffer contents are not valid.

```
        BIOS Function: ?MOV
        Move a block of bytes from one location in memory to another.
        Entry Parameters: HL=destination address
                          DE=source address
                          BC=byte count
        Returned  Values: HL and DE must point to next bytes following move operation
```

The BDOS calls the ?MOV routine to perform memory-to-memory block moves. This allows use of the Zilog Z-80 LDIR instruction or special DMA hardware, if available. Note that arguments in HL and DE are reversed from the Z-80 machine instruction, necessitating the use of XCHG instructions on either side of the LDIR. The BDOS uses this routine for all large memory copy operations. On return, the HL and DE registers are expected to point to the next bytes following the move.

```
        BIOS Function: ?DSCRF
        Perform direct screen functions.
        Entry Parameters: DE points to:
                  byte 0: Subfunction number
              bytes 1-2: Pointer to extended information
           or
                  byte 1: Column value
                  byte 2: Row value
        Returned  Values: Depends upon subfunction (described below)
```

The Direct Screen Function routines provide direct access to cursor movement and screen editing functions for video-intensive applications, such as word processing and electronic spreadsheets. Direct access is important in systems with memory-mapped displays. This call not only permits direct access to these functions, but can also return information to the calling program about whether a specific function executes quickly or slowly on a particular system. If a particular function is emulated by BIOS display drivers, the system response will be slower than the direct screen access.

Upon entry to this BIOS function, register DE points to a 3-byte block containing the following:

```
          byte 0: Subfunction number
       bytes 1-2: Pointer to extended information
   or
          byte 1: Column value
          byte 2: Row value
```

It is the responsibility of the BIOS to report in the bit-map returned by subfunction 0 whether the subfunction is supported. The subfunctions supported by ?DSCRF are described in the following table.

Table 4-4. Direct screen subfunctions

```
Format: Subfunction number -- Name in full
        Description, or returned value
```

0 -- Subfunctions supported
Returned value: HL=pointer to a 4-byte block of memory as follows:

```
        byte 0: 07 06 05 04 03 02 01 00
        byte 1: 15 14 13 12 11 10 09 08
        byte 2: 23 22 21 20 19 18 17 16
        byte 3:             27 26 25 24
```

The corresponding bit is set if a particular subfunction is supported in the BIOS.

1 -- Subfunctions emulated
Returned value: HL=pointer to a 4-byte block of memory as in Subfunction 0, above.

2 -- Display size

```
Returned value: H=number of columns (n-1)
                L=number of rows (n-1)
```

3 -- Identify terminal
Returned value: HL=pointer to an ASCII null-terminated (NULL = 00H) identifier string. For example, a DEC VT-52-type terminal would return the bytes: ESCape, '/', 'K', NULL.

4 -- Cursor up
Does not scroll screen down if the cursor is at the top of screen.

5 -- Cursor down
Does not scroll screen up if the cursor is at the bottom of screen.

6 -- Cursor left
Wrap depends on the mode set by Subfunction 26 or 27.

7 -- Cursor right
Wrap depends on the mode set by Subfunction 26 or 27.

8 -- Cursor home
Move the cursor to the top left corner of screen.

9 -- Cursor on
Make the cursor visible.

10 -- Cursor off
Make the cursor invisible.

11 -- Direct cursor addressing
Move the cursor to absolute column and row indicated by the second and third bytes pointed to by DE upon entry to the ?DSCRF function.

12 -- Clear display
Move the cursor to the top left corner of the screen, and erase the screen.

13 -- Erase to end of line
Erase all characters to the right of the cursor.

14 -- Erase to end of screen
Erase all characters to the right of the cursor to the end of the screen.

15 -- Enter ANSI mode

Place the display hardware in the ANSI mode.

16 -- Enter VT-52 mode
Place the display hardware in the VT-52 mode.

17 * -- Enter graphics mode
Place the display hardware in the graphics mode.

18 * -- Exit graphics mode
Return the display hardware to the current terminal mode, either ANSI or VT-52.

19 * -- Enter alternate keypad mode

20 * -- Exit alternate keypad mode

21 -- Enter hold screen mode

22 -- Exit hold screen mode

23 -- Enter reverse video mode

24 -- Exit reverse video mode

25 -- Reverse line-feed

26 -- Enable wrap-around at end of line

27 -- Truncate characters at end of line

(* = Not supported by the Sharp MZ-800 Personal CP/M.)

```
        BIOS Function: ?BYTBC
        "Byte BLT Copy".
        Entry Parameters: DE=BCB address
        Returned  Values: A=00H if implemented, or
                          A=0FFH if not implemented
```

ROCHE> BCB = Byte Control Block. This is all I know...

```
        BIOS Function: ?BYTBA
        "Byte BLT Alter".
        Entry Parameters: DE=BCB address
        Returned  Values: A=00H if implemented, or
                          A=0FFH if not implemented
```

ROCHE> BCB = Byte Control Block. This is all I know...

Section 5: Disk definition information

5.1 Introduction

The BIOS provides a standard interface to the physical Input/Output devices in your system. The BIOS interface is defined by the functions described in Section 4. Those functions, taken together, constitute a model of the hardware environment. Each BIOS is responsible for mapping that model onto the real hardware.

In addition, the BIOS contains disk definition tables which define the characteristics of the disk devices which are present, and provides some storage for use by the BDOS in maintaining disk directory information.

Section 4 describes the functions that must be performed by the BIOS, and the external interface to those functions. This section contains additional information describing the structure and significance of the disk definition tables, and information about sector blocking and deblocking. Careful choices of disk parameters and disk buffering methods are necessary, if you are to achieve the best possible performance from Personal CP/M. Therefore, this section should be read thoroughly before writing a custom BIOS.

5.2 Disk definition tables

As in other CP/M systems, Personal CP/M uses a set of tables to define disk device characteristics. This section describes each of these tables, and discusses choices of certain parameters.

5.2.1 Disk Parameter Header

Each disk drive has an associated 16-byte Disk Parameter Header (DPH) that contains information about the disk drive, and also provides a scratch area for certain BDOS operations. Each drive must have its own unique DPH. The format of a Disk Parameter Header is shown in Figure 5-1.

```
XLT  0000  0000  0000  DIRBUF  DPB  CSV  ALV
16b  16b   16b   16b    16b    16b  16b  16b
```

Figure 5-1. Disk Parameter Header

Each element of the DPH is a word (16-bit) value, and is described in Table 5-1.

Table 5-1. Disk Parameter Header elements

Format: Address
Description

XLT
Address of the logical-to-physical sector translation table, if used for this particular drive. Otherwise, the value of XLT is 0000H if there is no translation table for this drive (that is to say: the physical and logical sector numbers are the same). Disk drives with identical sector translation can share the same translate table.

0000
Three scratch pad words for use within the BDOS. The initial value is unimportant.

DIRBUF
Address of a 128-byte scratch pad area for directory operations within BDOS. All DPHs address the same scratch pad area.

DPB
Address of a disk parameter block for this drive. Drives with identical disk characteristics can address the same disk parameter block.

CSV
Address of a scratch pad area used for software check for changed disks. This address is different for each DPH.

ALV
Address of a scratch pad area used by the BDOS to keep disk storage allocation information. This address is different for each DPH.

Given n disk drives, the DPHs are arranged in an array. The first row of 16 bytes corresponds to drive 0, with the last row corresponding to drive n-1. The array appears in Figure 5-2.

```
DPBASE:
     +---------+------+------+------+--------+----------+---------+---------+
  00 | XLT  00 | 0000 | 0000 | 0000 | DIRBUF | DPBB  00 | CSV  00 | ALV  00 |
     +---------+------+------+------+--------+----------+---------+---------+
  01 | XLT  01 | 0000 | 0000 | 0000 | DIRBUF | DPBB  01 | CSV  01 | ALV  01 |
     +---------+------+------+------+--------+----------+---------+---------+
 ... :         :      :      :      :        :          :         :         :
     +---------+------+------+------+--------+----------+---------+---------+
 n-1 | XLT n-1 | 0000 | 0000 | 0000 | DIRBUF | DPBB n-1 | CSV n-1 | ALV n-1 |
     +---------+------+------+------+--------+----------+---------+---------+
```

Figure 5-2. Array of DPH entries

The label DPBASE defines the base address of the DPH table.

A responsibility of the SETDSK subroutine is to return the base address of the DPH for the selected drive. The following sequence of operations returns the table address, with a 0000H returned if the selected drive does not exist.

```
ndisks  EQU     4               ; Number of disk drives
  ...
setdsk: ; Select disk given by BC
        LXI     H,0000H         ; Error code
        MOV     A,C             ; Drive Ok?
        CPI     ndisks          ; Carry if so
        RNC                     ; Return if error
        ; No error, continue.
        MOV     L,C             ; Low (disk)
        MOV     H,B             ; High (disk)
        DAD     H               ; *2
        DAD     H               ; *4
        DAD     H               ; *8
        DAD     H               ; *16
        LXI     D,dpbase        ; First DPH
        DAD     D               ; DPH (disk)
        RET
```

Figure 5-3. SETDSK example

The translation vectors (XLT 00 through XLT n-1) are located elsewhere in the BIOS, and simply correspond one-for-one with the logical sector numbers zero through the sector count 1.

5.2.2 Disk Parameter Block

The Disk Parameter Block (DPB), which is addressed by one or more DPHs, take this general form:

```
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
| SPT | BSH | BLM | EXM | DSM | DRM | AL0 | AL1 | CKS | OFF |
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
  16b    8b    8b    8b   16b   16b    8b    8b   16b   16b
```

where each is a byte or word value, as shown by the "8b" or "16b" indicator below the field. Table 5-2 gives a description of the Disk Parameter Block.

Table 5-2. Disk Parameter Block description

Format: Name
Description

Name	Description
SPT	Total number of sectors per track.
BSH	Data allocation block shift factor, determined by the data block allocation size.
BLM	Data allocation block mask ((2^BSH)-1).
EXM	Extent mask, determined by the data block allocation size and the number of disk blocks.
DSM	Total storage capacity of the disk drive.
DRM	Total number of directory entries that can be stored on this drive.
(AL0, AL1 determine reserved directory blocks.)	
CKS	Size of the directory check vector.
OFF	Number of reserved tracks at the beginning of the (logical) disk.

The values of BSH and BLM implicitly determine the data block allocation size, BLS, which is not an entry in the DPB. Given that the designer has selected a value for BLS, the values of BSH and BLM are shown in the following table:

BLS	BSH	BLM
1024	3	3
2048	4	15
4096	5	31
8192	6	63
16384	7	127

All values are decimal. The value of EXM depends upon both the BLS, and whether the DSM value is less than 256 or greater than 255. For DSM less than 256, the value of EXM is given by:

BLS	EXM
1024	0
2048	1
4096	3
8192	7
16384	15

For DSM greater than 255, the value of EXM is given by:

BLS	EXM
1024	N/A
2048	0
4096	1
8192	3
16384	7

The value of DSM is the maximum data block number measured in BLS units supported by this particular drive. The product BLS * (DSM + 1) is the total number of bytes held by the drive and, of course, must be within the capacity of the physical disk, not counting the reserved operating system tracks.

The DRM entry is one less than the total number of directory entries that can take on a 16-bit value. The values of AL0 and AL1 are determined by DRM. AL0 and AL1 values together can be considered a string of 16-bits, as shown below:

```
        +-----------------------+-----------------------+
        |          AL0          |          AL1          |
        +--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
        |00|01|02|03|04|05|06|07|08|09|10|11|12|13|14|15|
        +--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
```

Position 00 corresponds to the high-order bit of the byte AL0, and position 15 corresponds to the low-order bit of the byte AL1. Each bit position reserves a data block for a number of directory entries, thus allowing a total of 16 data blocks to be assigned for directory entries (bits are assigned starting at 00, and filled to the right through position 15). Each directory entry occupies 32 bytes, resulting in the following table:

BLS	Directory entries
1024	32 times # bits
2048	64 times # bits
4096	128 times # bits
8192	256 times # bits
16384	512 times # bits

If DRM = 127 (128 directory entries) and BLS = 1024, there are 32 directory entries per block, requiring 4 reserved blocks. In this case, the 4 high-order bits of AL0 are set, resulting in the values AL0 = 0F0H and AL1 = 00H.

The CKS value is determined as follows:

1. If the disk drive media is removable, then CKS = (DRM + 1) / 4, where DRM is the last directory entry number.

2. If the media is fixed, then CKS = 0. No directory records are checked in this case.

Finally, the OFF field determines the number of tracks that are skipped at the beginning of the physical disk. This value is automatically added whenever SETTRK is called. It can be used as a mechanism for skipping reserved operating system tracks, or for partitioning a large disk into smaller segmented sections.

To complete the discussion of the DPB, several DPHs can address the same DPB if their drive characteristics are identical. Further, the DPB can be dynamically changed when a new drive is addressed. Since the BDOS copies the DPB values to a local area whenever the SELDSK function is called, simply change the pointer in the DPH.

Returning back to the DPH for a particular drive, the two address values, CSV and ALV, reference areas of uninitialized memory in the BIOS data segment. The areas must be unique for each drive, and the size of each area is determined by the values in the DPB.

The size of the area addressed by CSV is CKS bytes, which is sufficient to hold the directory check information for this particular drive. If the media is removable, you must reserve (DRM + 1) / 4 bytes for directory check use. If the media is fixed, no storage is reserved.

The size of the area addressed by ALV is determined by the maximum number of data blocks allowed for this particular disk, and is equal to 2*(DSM/8+1). Two copies of the allocation map for the disk are kept in this area: the first vector stores temporarily-allocated blocks resulting from WRITE operations, the second stores permanently-allocated blocks resulting from CLOSE FILE operations.

5.3 The DISKDEF macro library

A macro library which is on the distribution disk, called DISKDEF, greatly simplifies the table construction process. Of course, you must have access to the MAC macro-assembler to use the DISKDEF facility.

A BIOS disk definition consists of the following sequence of macro statements:

```
        MACLIB  DISKDEF
        ...
        DISKS   n
        DISKDEF 0,...
        DISKDEF 1,...
        ...
        DISKDEF n-1,...
        ENDEF
```

The MACLIB statement loads the DISKDEF.LIB file (on the same disk as the BIOS) into MAC's internal tables. The DISKS macro call follows, which specifies the number of drives to be configured with the user's system, where n is an integer from 1 to 16. A series of DISKDEF macro calls then follow, that define the characteristics of each logical disk, 0 through n-1 (corresponding to logical drives A through P). The DISKS and DISKDEF macros generate the in-line fixed data tables described in the previous section, and must be placed in a non-executable portion of the BIOS, typically directly following the BIOS jump vector.

The remaining portion of the BIOS is defined following the DISKDEF macros, with the ENDEF macro call immediately preceding the END statement. The ENDEF (End of diskDEF) macro generates the necessary uninitialized RAM areas, which are located in memory above the BIOS.

The form of the DISKDEF macro call is as follows:

```
        DISKDEF dn,fsc,lsc,[skf],bls,dks,dir,cks,cks,ofs,[0]
where
        dn  is the logical disk number, 0 to n-1
        fsc is the first physical sector number (0 or 1)
        lsc is the last sector number
        skf is the optional sector skew factor
        bls is the data block allocation size
        dks is the number of blocks on the disk
        dir is the number of directory entries
        cks is the number of "checked" directory entries
        ofs is the track offset to logical track zero
        [0] is an optional CP/M 1.4 compatibility flag
```

The value DN is the drive number being defined with this DISKDEF macro invocation.

Parameter FSC accounts for differing sector numbering systems, and is usually zero or one.

The last numbered sector on the track is defined by LSC.

When present, the SKF parameter defines the sector skew factor, which is used to create a sector translation table according to the skew. If the number of sectors is less than 256, a 1-byte table is created. Otherwise, each translation table element occupies 2 bytes. No sector translation table is created if the SKF parameter is either omitted or equal to zero.

The BLS parameter specifies the number of bytes allocated to each data block, and takes on the values 1024, 2048, 4096, 8192, or 16384. Generally, performance increases with larger data block sizes, since there are fewer directory references, and logically-connected data records are physically close on the disk. Also, each directory entry addresses more data, and the BIOS-resident data space is reduced.

The DKS parameter specifies the total disk size in BLS units. That is to say: if the BLS = 2048 and DKS = 1000, the total disk capacity is 2,048,000 bytes. If DKS is greater than 255, the block size parameter BLS must be greater than 1024.

The value of DIR is the total number of directory entries, which may exceed 255, if desired.

The CKS parameter determines the number of directory items to check on each directory scan. It is used internally to detect changed disks during system operation, where an intervening cold or warm boot has not occurred. When a

disk is removed, Personal CP/M automatically marks the disk as Read-Only. As mentioned earlier, the value of CKS=DIR when the medium is easily changed, as in a floppy disk subsystem. If the disk is fixed, the value of CKS is typically zero, since the probability of changing disks without a warm start is low.

The value of OFS determines the number of tracks to skip when this particular drive is addressed. This permits reserving a number of tracks for the operating system, or for simulating a number of drives on a single large-capacity physical drive.

Finally, the [0] parameter is included when file compatibility is required with versions of CP/M 1.4 that have been modified for higher density disks. This parameter ensures that only 16 kilobytes is allocated for each directory record, as was the case for earlier CP/M versions. Normally, this parameter is not included.

For convenience and economy of table space, the special form:

```
        DISKDEF i,j
```

gives disk I the same characteristics as a previously-defined drive J. A common 4-drive, single-sided, single-density 8-inch disk system, which is compatible with CP/M 1.4, is defined using the following macro invocations:

```
        DISKS    4
        DISKDEF 0,1,26,6,1024,243,64,64,2
        DISKDEF 1,0
        DISKDEF 2,0
        DISKDEF 3,0
        ENDEF
```

with all disks having the same parameter values of 26 sectors for each track (numbered 1 through 26), 6 sectors skipped between each access, 1024 bytes for each data block, 243 data blocks, for a total of 243 kilobyte disk capacity, 64 checked directory entries, and 2 operating system tracks.

The DISKS macro generates n DPHs, starting at the DPH table address DPBASE generated by the macro. Each disk header block contains 16 bytes, as described earlier, and correspond one-for-one to each of the defined drives. For example, in a 4-drive system, the DISKS macro generates a table of the form:

```
        DPBASE  EQU     $
        DPE0:   DW      XLT0,0000H,0000H,0000H,DIRBUF,DPB0,CSV0,ALV0
        DPE1:   DW      XLT1,0000H,0000H,0000H,DIRBUF,DPB1,CSV1,ALV1
        DPE2:   DW      XLT2,0000H,0000H,0000H,DIRBUF,DPB2,CSV2,ALV2
        DPE3:   DW      XLT3,0000H,0000H,0000H,DIRBUF,DPB3,CSV3,ALV3
```

where the DPH labels are included for reference purposes, to show the beginning table address for each drive, zero through three. The values contained within the DPH are described in detail in Section 5.2.1. The check and allocation vector addresses are generated by the ENDEF macro in the RAM area following the BIOS code and tables.

You should note that, if the SKF (sector skew factor) parameter is omitted (or equal to zero), the translation table is omitted, and a 0000H value is inserted in the XLT position of the DPH for the disk. In a subsequent call to perform the logical-to-physical sector translation, SECTRN receives a translation table address of DE=0000H, and simply returns the original logical sector from BC in the HL register pair. A translate table is constructed when the SKF parameter is present, and the (non-zero) table address is placed into the corresponding DPHs. For example, the following table is constructed when the standard skew factor (skf = 6) is specified in the DISKDEF macro call:

```
        XLT0:   DB      1,7,13,19,25,5,11,17,23,3,9,15,21
                DB      2,8,14,20,26,6,12,18,24,4,10,16,22
```

Following the ENDEF macro call, a number of uninitialized data areas are defined. These data areas need not be a part of the BIOS that is loaded upon cold start, but must be available between the BIOS and the end of memory. The size of the uninitialized RAM area is determined by EQU statements generated by the ENDEF macro. For a standard 4-drive system, the ENDEF macro might produce this:

```
      4C72 =    BEGDAT  EQU     $
                (data areas)
      4DB0 =    ENDDAT  EQU     $
      013C =    DATSIZ  EQU     $-BEGDAT
```

which indicates that uninitialized RAM begins at location 4C72H, ends at 4DB0H-1, and occupies 013CH bytes. You must ensure that these addresses are free for use after the system is loaded.

After modification, you can utilize the STAT program to check drive characteristics, because STAT uses the disk parameter block to decode the drive information:

```
        A>stat x:dsk:
```

This command decodes the disk parameter block for drive specifier X (A through P), and displays the following values:

```
        r: 128-byte record capacity
        k: kilobyte drive capacity
        d: 32-byte directory entries
        c: checked directory entries
        e: records/extent
        b: records/block
        s: records/track
        t: reserved tracks
```

Three examples of DISKDEF macro invocations are shown below, with corresponding STAT parameter values. The last example produces an 8-Megabyte system:

```
                 DISKDEF 0,1,58,,2048,256,128,128,2
        r=4096,  k=512, d=128, c=128, e=256, b=16, s=58, t=2

                 DISKDEF 0,1,58,,2048,300,0,2
        r=16384, k=2048, d=300, c=0, e=128, b=16, s=58, t=2

                 DISKDEF 0,1,58,,16384,512,128,128,2
        r=65535, k=8192, d=128, c=128, e=1024, b=128, s=58, t=2
```

5.4 Sector blocking and deblocking

Upon each call to the BIOS WRITE function, the Personal CP/M BDOS includes information that allows effective sector blocking and deblocking, where the disk subsystem has a sector size that is a multiple of the basic 128-byte unit. The purpose, here, is to present a general-purpose algorithm that can be included within the BIOS, and that uses the BDOS information to perform the operations automatically.

On each call to WRITE, the BDOS provides the following information in register C:

```
        0=normal sector write
        1=write to directory sector
        2=write to the first sector of a new data block
```

Condition zero occurs whenever the next WRITE operation is into a previously written area, such as a random mode record update, or when the WRITE is to other than the first sector of an unallocated block, or when the WRITE is not into the directory area.

Condition one occurs when a WRITE into the directory area is performed.

Condition two occurs when the first record (only) of a newly-allocated data block is written. In most cases, application programs read or write multiple 128-byte sectors in sequence; there is little overhead involved in either operation when blocking or deblocking records, since preread operations can be avoided when writing record.

Index

```
  (To be done by WS4...)

  EOF
```