DIGITAL RESEARCH

Post Office Box 579, Pacific Grove, California 93950, (408) 649-3896

CP/M 2.0 ALTERATION GUIDE

Copyright

Disclaimer

Trademarks

DIGITAL RESEARCH

Post Office Box 579, Pacific Grove, California 93950, (408) 649-3896

CP/M 2.0 ALTERATION GUIDE

CP/M 2.Ø ALTERATION GUIDE

1. INTRODUCTION

The standard CP/M system assumes operation on an Intel MDS-800 microcomputer development system, but is designed so that the user can alter a specific set of subroutines which define the hardware operating environment. In this way, the user can produce a diskette which operates with any IBM-3741 format compatible drive controller and other peripheral devices.

Although standard CP/M 2.0 is configured for single density floppy disks, field-alteration features allow adaptation to a wide variety of disk subsystems from single drive minidisks through high-capacity "hard disk" systems. In order to simplify the following adaptation process, we assume that CP/M 2.0 will first be configured for single density floppy disks where minimal editing and debugging tools are available. If an earlier version of CP/M is available, the customizing process is eased considerably. In this latter case, you may wish to briefly review the system generation process, and skip to later sections which discuss system alteration for non-standard disk systems.

In order to achieve device independence, CP/M is separated into three distinct modules:

```
BIOS - basic I/O system which is environment dependent
BDOS - basic disk operating system which is not dependent
       upon the hardware configuration
CCP  - the console command processor which uses the BDOS
```

Of these modules, only the BIOS is dependent upon the particular hardware. That is, the user can "patch" the distribution version of CP/M to provide a new BIOS which provides a customized interface between the remaining CP/M modules and the user's own hardware system. The purpose of this document is to provide a step-by-step procedure for patching your new BIOS into CP/M.

If CP/M is being tailored to your computer system for the first time, the new BIOS requires some relatively simple software development and testing. The standard BIOS is listed in Appendix B, and can be used as a model for the customized package. A skeletal version of the BIOS is given in Appendix C which can serve as the basis for a modified BIOS. In addition to the BIOS, the user must write a simple memory loader, called GETSYS, which brings the operating system into memory. In order to patch the new BIOS into CP/M, the user must write the reverse of GETSYS, called PUTSYS, which places an altered version of CP/M back onto the diskette. PUTSYS can be derived from GETSYS by changing the disk read commands into disk write commands. Sample skeletal GETSYS and PUTSYS programs are described in Section 3, and listed in Appendix D. In order to make the CP/M system work automatically, the user must also supply a cold start loader, similar to the one provided with CP/M (listed in Appendices A and B). A skeletal form of a cold start loader is given in Appendix E which can serve as a model for your loader.

2. FIRST LEVEL SYSTEM REGENERATION

The procedure to follow to patch the CP/M system is given below in several steps. Address references in each step are shown with a following "H" which denotes the hexadecimal radix, and are given for a 20K CP/M system. For larger CP/M systems, add a "bias" to each address which is shown with a "+b" following it, where b is equal to the memory size - 20K. Values for b in various standard memory sizes are

24K:	b = 24K - 20K = 4K = 1000H
32K:	b = 32K - 20K = 12K = 3000H
40K:	b = 40K - 20K = 20K = 5000H
48K:	b = 48K - 20K = 28K = 7000H
56K:	b = 56K - 20K = 36K = 9000H
62K:	b = 62K - 20K = 42K = A800H
64K:	b = 64K - 20K = 44K = B000H

Note: The standard distribution version of CP/M is set for operation within a 20K memory system. Therefore, you must first bring up the 20K CP/M system, and then configure it for your actual memory size (see Second Level System Generation).

(1) Review Section 4 and write a GETSYS program which reads the first two tracks of a diskette into memory. The data from the diskette must begin at location 3380H. Code GETSYS so that it starts at location 100H (base of the TPA), as shown in the first part of Appendix d.

(2) Test the GETSYS program by reading a blank diskette into memory, and check to see that the data has been read properly, and that the diskette has not been altered in any way by the GETSYS program.

(3) Run the GETSYS program using an initialized CP/M diskette to see if GETSYS loads CP/M starting at 3380H (the operating system actually starts 128 bytes later at 3400H).

(4) Review Section 4 and write the PUTSYS program which writes memory starting at 3380H back onto the first two tracks of the diskette. The PUTSYS program should be located at 200H, as shown in the second part of Appendix D.

(5) Test the PUTSYS program using a blank uninitialized diskette by writing a portion of memory to the first two tracks; clear memory and read it back using GETSYS. Test PUTSYS completely, since this program will be used to alter CP/M on disk.

(6) Study Sections 5, 6, and 7, along with the distribution version of the BIOS given in Appendix B, and write a simple version which performs a similar function for the customized environment. Use the program given in Appendix C as a model. Call this new BIOS by the name CBIOS (customized BIOS). Implement only the primitive disk operations on a single drive, and simple console input/output functions in this phase.

(7) Test CBIOS completely to ensure that it properly performs console character I/O and disk reads and writes. Be especially careful to ensure that no disk write operations occur accidently during read operations, and check that the proper track and sectors are addressed on all reads and writes. Failure to make these checks may cause destruction of the initialized CP/M system after it is patched.

(8) Referring to Figure 1 in Section 5, note that the BIOS is placed between locations 4A00H and 4FFFH. Read the CP/M system using GETSYS and replace the BIOS segment by the new CBIOS developed in step (6) and tested in step (7). This replacement is done in the memory of the machine, and will be placed on the diskette in the next step.

(9) Use PUTSYS to place the patched memory image of CP/M onto the first two tracks of a blank diskette for testing.

(10) Use GETSYS to bring the copied memory image from the test diskette back into memory at 3380H, and check to ensure that it has loaded back properly (clear memory, if possible, before the load). Upon successful load, branch to the cold start code at location 4A00H. The cold start routine will initialize page zero, then jump to the CCP at location 3400H which will call the BDOS, which will call the CBIOS. The CBIOS will be asked by the CCP to read sixteen sectors on track 2, and if successful, CP/M will type "A>", the system prompt.

When you make it this far, you are almost on the air. If you have trouble, use whatever debug facilities you have available to trace and breakpoint your CBIOS.

(11) Upon completion of step (10), CP/M has prompted the console for a command input. Test the disk write operation by typing

    SAVE 1 X.COM

(recall that all commands must be followed by a carriage return).

CP/M should respond with another prompt (after several disk accesses):

    A>

If it does not, debug your disk write functions and retry.

(12) Then test the directory command by typing

    DIR

CP/M should respond with

    A: X   COM

(13) Test the erase command by typing

    ERA X.COM

CP/M should respond with the A prompt. When you make it this far, you should have an operational system which will only require a bootstrap loader to function completely.

(14) Write a bootstrap loader which is similar to GETSYS, and place it on track Ø, sector 1 using PUTSYS (again using the test diskette, not the distribution diskette). See Sections 5 and 8 for more information on the bootstrap operation.

(15) Retest the new test diskette with the bootstrap loader installed by executing steps (11), (12), and (13). Upon completion of these tests, type a control-C (control and C keys simultaneously). The system should then execute a "warm start" which reboots the system, and types the A prompt.

(16) At this point, you probably have a good version of your customized CP/M system on your test diskette. Use GETSYS to load CP/M from your test diskette. Remove the test diskette, place the distribution diskette (or a legal copy) into the drive, and use PUTSYS to replace the distribution version by your customized version. Do not make this replacement if you are unsure of your patch since this step destroys the system which was sent to you from Digital Research.

(17) Load your modified CP/M system and test it by typing

    DIR

CP/M should respond with a list of files which are provided on the initialized diskette. One such file should be the memory image for the debugger, called DDT.COM.

NOTE: from now on, it is important that you always reboot the CP/M system (ctl-C is sufficient) when the diskette is removed and replaced by another diskette, unless the new diskette is to be read only.

(18) Load and test the debugger by typing

    DDT

(see the document "CP/M Dynamic Debugging Tool (DDT)" for operating procedures. You should take the time to become familiar with DDT, it will be your best friend in later steps.

(19) Before making further CBIOS modifications, practice using the editor (see the ED user's guide), and assembler (see the ASM user's guide). Then recode and test the GETSYS, PUTSYS, and CBIOS programs using ED, ASM, and DDT. Code and test a COPY program which does a sector-to-sector copy from one diskette to another to obtain back-up copies of the original diskette (NOTE: read your CP/M Licensing Agreement; it specifies your legal responsibilities when copying the CP/M system). Place the copyright notice

Copyright (c), 1979
Digital Research

on each copy which is made with your COPY program.

(20) Modify your CBIOS to include the extra functions for punches, readers, signon messages, and so-forth, and add the facilities for a additional disk drives, if desired. You can make these changes with the GETSYS and PUTSYS programs which you have developed, or you can refer to the following section, which outlines CP/M facilities which will aid you in the regeneration process.

You now have a good copy of the customized CP/M system. Note that although the CBIOS portion of CP/M which you have developed belongs to you, the modified version of CP/M which you have created can be copied for your use only (again, read your Licensing Agreement), and cannot be legally copied for anyone else's use.

It should be noted that your system remains file-compatible with all other CP/M systems, (assuming media compatiblity, of course) which allows transfer of non-proprietary software between users of CP/M.

3. SECOND LEVEL SYSTEM GENERATION

Now that you have the CP/M system running, you will want to configure CP/M for your memory size. In general, you will first get a memory image of CP/M with the "MOVCPM" program (system relocator) and place this memory image into a named disk file. The disk file can then be loaded, examined, patched, and replaced using the debugger, and system generation program. For further details on the operation of these programs, see the "Guide to CP/M Features and Facilities" manual.

Your CBIOS and BOOT can be modified using ED, and assembled using ASM, producing files called CBIOS.HEX and BOOT.HEX, which contain the machine code for CBIOS and BOOT in Intel hex format.

To get the memory image of CP/M into the TPA configured for the desired memory size, give the command:

```
MOVCPM xx *
```

where "xx" is the memory size in decimal K bytes (e.g., 32 for 32K). The response will be:

```
CONSTRUCTING xxK CP/M VERS 2.0
READY FOR "SYSGEN" OR
"SAVE 34 CPMxx.COM"
```

At this point, an image of a CP/M in the TPA configured for the requested memory size. The memory image is at location 0900H through 227FH. (i.e., The BOOT is at 0900H, the CCP is at 980H, the BDOS starts at 1180H, and the BIOS is at 1F80H.) Note that the memory image has the standard MDS-800 BIOS and BOOT on it. It is now necessary to save the memory image in a file so that you can patch your CBIOS and CBOOT into it:

```
SAVE 34 CPMxx.COM
```

The memory image created by the "MOVCPM" program is offset by a negative bias so that it loads into the free area of the TPA, and thus does not interfere with the operation of CP/M in higher memory. This memory image can be subsequently loaded under DDT and examined or changed in preparation for a new generation of the system. DDT is loaded with the memory image by typing:

```
DDT CPMxx.COM                   Load DDT, then read the CPM
                                image
```

DDT should respond with

```
NEXT  PC
2300  0100
-                               (The DDT prompt)
```

You can then use the display and disassembly commands to examine portions of the memory image between 900H and 227FH. Note, however, that to find any particular address within the memory image, you must apply the negative bias to the CP/M address to find the actual address. Track 00, sector 01 is loaded to location 900H (you should find the cold start loader at 900H to 97FH), track 00, sector 02 is loaded into 980H (this is the base of the CCP), and so-forth through the entire CP/M system load. In a 20K system, for example, the CCP resides at the CP/M address 3400H, but is placed into memory at 980H by the SYSGEN program. Thus, the negative bias, denoted by n, satisfies

```
3400H + n = 980H, or n = 980H - 3400H
```

Assuming two's complement arithmetic, n = D580H, which can be checked by

```
3400H + D580H = 10980H = 0980H (ignoring high-order
                                overflow).
```

Note that for larger systems, n satisfies

```
(3400H+b) + n = 980H, or
n = 980H - (3400H + b), or
n = D580H - b.
```

The value of n for common CP/M systems is given below

memory size	bias b	negative offset n
20K	0000H	D580H - 0000H = D580H
24K	1000H	D580H - 1000H = C580H
32K	3000H	D580H - 3000H = A580H
40K	5000H	D580H - 5000H = 8580H
48K	7000H	D580H - 7000H = 6580H
56K	9000H	D580H - 9000H = 4580H
62K	A800H	D580H - A800H = 2D80H
64K	B000H	D580H - B000H = 2580H

Assume, for example, that you want to locate the address x within the memory image loaded under DDT in a 20K system. First type

```
Hx,n        Hexadecimal sum and difference
```

and DDT will respond with the value of x+n (sum) and x-n (difference). The first number printed by DDT will be the actual memory address in the image where the data or code will be found. The input

```
H3400,D580
```

for example, will produce 980H as the sum, which is where the CCP is located in the memory image under DDT.

Use the L command to disassemble portions the BIOS located at (4A00H+b)-n which, when you use the H command, produces an actual address of 1F80H. The disassembly command would thus be

```
        L1F80
```

It is now necessary to patch in your CBOOT and CBIOS routines. The BOOT resides at location 0900H in the memory image. If the actual load address is "n", then to calculate the bias (m) use the command:

```
        H900,n                  Subtract load address from
                                target address.
```

The second number typed in response to the command is the desired bias (m). For example, if your BOOT executes at 0080H, the command:

```
        H900,80
```

will reply

```
        0980 0880               Sum and difference in hex.
```

Therefore, the bias "m" would be 0880H. To read-in the BOOT, give the command:

```
        ICBOOT.HEX              Input file CBOOT.HEX
```

Then:

```
        Rm                      Read CBOOT with a bias of
                                m (=900H-n)
```

You may now examine your CBOOT with:

```
        L900
```

We are now ready to replace the CBIOS. Examine the area at 1F80H where the original version of the CBIOS resides. Then type

```
        ICBIOS.HEX              Ready the "hex" file for loading
```

assume that your CBIOS is being integrated into a 20K CP/M system, and thus is origined at location 4A00H. In order to properly locate the CBIOS in the memory image under DDT, we must apply the negative bias n for a 20K system when loading the hex file. This is accomplished by typing

```
        RD580                   Read the file with bias D580H
```

Upon completion of the read, re-examine the area where the CBIOS has been loaded (use an "L1F80" command), to ensure that is was loaded properly. When you are satisfied that the change has been made, return from DDT using a control-C or "G0" command.

Now use SYSGEN to replace the patched memory image back onto a diskette (use a test diskette until you are sure of your patch), as shown in the following interaction

```
SYSGEN                        Start the SYSGEN program
SYSGEN VERSION 2.Ø            Sign-on message from SYSGEN
SOURCE DRIVE NAME (OR RETURN TO SKIP)
                              Respond with a carriage return
                              to skip the CP/M read operation
                              since the system is already in
                              memory.
DESTINATION DRIVE NAME (OR RETURN TO REBOOT)
                              Respond with "B" to write the
                              new system to the diskette in
                              drive B.
DESTINATION ON B, THEN TYPE RETURN
                              Place a scratch diskette in
                              drive B, then type return.
FUNCTION COMPLETE
DESTINATION DRIVE NAME (OR RETURN TO REBOOT)
```

Place the scratch diskette in your drive A, and then perform a coldstart to bring up the new CP/M system you have configured.

Test the new CP/M system, and place the Digital Research copyright notice on the diskette, as specified in your Licensing Agreement:

Copyright (c), 1979
Digital Research

4. SAMPLE GETSYS AND PUTSYS PROGRAMS

The following program provides a framework for the GETSYS and PUTSYS programs referenced in Section 2. The READSEC and WRITESEC subroutines must be inserted by the user to read and write the specific sectors.

```
;   GETSYS PROGRAM - READ TRACKS 0 AND 1 TO MEMORY AT 3380H
;   REGISTER                USE
;      A              (SCRATCH REGISTER)
;      B              TRACK COUNT (0, 1)
;      C              SECTOR COUNT (1,2,...,26)
;      DE             (SCRATCH REGISTER PAIR)
;      HL             LOAD ADDRESS
;      SP             SET TO STACK ADDRESS
;
START: LXI    SP,3380H      ;SET STACK POINTER TO SCRATCH AREA
       LXI    H, 3380H      ;SET BASE LOAD ADDRESS
       MVI    B, 0          ;START WITH TRACK 0
RDTRK:                      ;READ NEXT TRACK (INITIALLY 0)
       MVI    C,1           ;READ STARTING WITH SECTOR 1
RDSEC:                      ;READ NEXT SECTOR
       CALL   READSEC       ;USER-SUPPLIED SUBROUTINE
       LXI    D,128         ;MOVE LOAD ADDRESS TO NEXT 1/2 PAGE
       DAD    D             ;HL = HL + 128
       INR    C             ;SECTOR = SECTOR + 1
       MOV    A,C           ;CHECK FOR END OF TRACK
       CPI    27
       JC     RDSEC         ;CARRY GENERATED IF SECTOR < 27
;
;   ARRIVE HERE AT END OF TRACK, MOVE TO NEXT TRACK
       INR    B
       MOV    A,B           ;TEST FOR LAST TRACK
       CPI    2
       JC     RDTRK         ;CARRY GENERATED IF TRACK < 2
;
;   ARRIVE HERE AT END OF LOAD, HALT FOR NOW
       HLT
;
;   USER-SUPPLIED SUBROUTINE TO READ THE DISK
READSEC:
;   ENTER WITH TRACK NUMBER IN REGISTER B,
;         SECTOR NUMBER IN REGISTER C, AND
;         ADDRESS TO FILL IN HL
;
       PUSH   B             ;SAVE B AND C REGISTERS
       PUSH   H             ;SAVE HL REGISTERS
       .........................................
       perform disk read at this point, branch to

       label START if an error occurs
       .........................................
       POP    H             ;RECOVER HL
       POP    B             ;RECOVER B AND C REGISTERS
       RET                  ;BACK TO MAIN PROGRAM

       END    START
```

Note that this program is assembled and listed in Appendix C for reference purposes, with an assumed origin of 100H. The hexadecimal operation codes which are listed on the left may be useful if the program has to be entered through your machine's front panel switches.

The PUTSYS program can be constructed from GETSYS by changing only a few operations in the GETSYS program given above, as shown in Appendix D. The register pair HL become the dump address (next address to write), and operations upon these registers do not change within the program. The READSEC subroutine is replaced by a WRITESEC subroutine which performs the opposite function: data from address HL is written to the track given by register B and sector given by register C. It is often useful to combine GETSYS and PUTSYS into a single program during the test and development phase, as shown in the Appendix.

5. DISKETTE ORGANIZATION

The sector allocation for the standard distribution version of CP/M is given here for reference purposes. The first sector (see table on the following page) contains an optional software boot section. Disk controllers are often set up to bring track Ø, sector 1 into memory at a specific location (often location ØØØØH). The program in this sector, called BOOT, has the responsibility of bringing the remaining sectors into memory starting at location 34ØØH+b. If your controller does not have a built-in sector load, you can ignore the program in track Ø, sector 1, and begin the load from track Ø sector 2 to location 34ØØH+b.

As an example, the Intel MDS-8ØØ hardware cold start loader brings track Ø, sector 1 into absolute address 3ØØØH. Upon loading this sector, control transfers to location 3ØØØH, where the bootstrap operation commences by loading the remainder of tracks Ø, and all of track 1 into memory, starting at 34ØØH+b. The user should note that this bootstrap loader is of little use in a non-MDS environment, although it is useful to examine it since some of the boot actions will have to be duplicated in your cold start loader.

Track#	Sector#	Page#	Memory Address	CP/M Module name
00	01		(boot address)	Cold Start Loader
00	02	00	3400H+b	CCP
"	03	"	3480H+b	"
"	04	01	3500H+b	"
"	05	"	3580H+b	"
"	06	02	3600H+b	"
"	07	"	3680H+b	"
"	08	03	3700H+b	"
"	09	"	3780H+b	"
"	10	04	3800H+b	"
"	11	"	3880H+b	"
"	12	05	3900H+b	"
"	13	"	3980H+b	"
"	14	06	3A00H+b	"
"	15	"	3A80H+b	"
"	16	07	3B00H+b	"
00	17	"	3B80H+b	CCP
00	18	08	3C00H+b	BDOS
"	19	"	3C80H+b	"
"	20	09	3D00H+b	"
"	21	"	3D80H+b	"
"	22	10	3E00H+b	"
"	23	"	3E80H+b	"
"	24	11	3F00H+b	"
"	25	"	3F80H+b	"
"	26	12	4000H+b	"
01	01	"	4080H+b	"
"	02	13	4100H+b	"
"	03	"	4180H+b	"
"	04	14	4200H+b	"
"	05	"	4280H+b	"
"	06	15	4300H+b	"
"	07	"	4380H+b	"
"	08	16	4400H+b	"
"	09	"	4480H+b	"
"	10	17	4500H+b	"
"	11	"	4580H+b	"
"	12	18	4600H+b	"
"	13	"	4680H+b	"
"	14	19	4700H+b	"
"	15	"	4780H+b	"
"	16	20	4800H+b	"
"	17	"	4880H+b	"
"	18	21	4900H+b	"
01	19	"	4980H+b	BDOS
01	20	22	4A00H+b	BIOS
"	21	"	4A80H+b	"
"	23	23	4B00H+b	"
"	24	"	4B80H+b	"
"	25	24	4C00H+b	"
01	26	"	4C80H+b	BIOS
02-76	01-26			(directory and data)

6. THE BIOS ENTRY POINTS

The entry points into the BIOS from the cold start loader and BDOS are detailed below. Entry to the BIOS is through a "jump vector" located at 4A00H+b, as shown below (see Appendices B and C, as well). The jump vector is a sequence of 17 jump instructions which send program control to the individual BIOS subroutines. The BIOS subroutines may be empty for certain functions (i.e., they may contain a single RET operation) during regeneration of CP/M, but the entries must be present in the jump vector.

The jump vector at 4A00H+b takes the form shown below, where the individual jump addresses are given to the left:

```
4A00H+b    JMP BOOT       ; ARRIVE HERE FROM COLD START LOAD
4A03H+b    JMP WBOOT      ; ARRIVE HERE FOR WARM START
4A06H+b    JMP CONST      ; CHECK FOR CONSOLE CHAR READY
4A09H+b    JMP CONIN      ; READ CONSOLE CHARACTER IN
4A0CH+b    JMP CONOUT     ; WRITE CONSOLE CHARACTER OUT
4A0FH+b    JMP LIST       ; WRITE LISTING CHARACTER OUT
4A12H+b    JMP PUNCH      ; WRITE CHARACTER TO PUNCH DEVICE
4A15H+b    JMP READER     ; READ READER DEVICE
4A18H+b    JMP HOME       ; MOVE TO TRACK 00 ON SELECTED DISK
4A1BH+b    JMP SELDSK     ; SELECT DISK DRIVE
4A1EH+b    JMP SETTRK     ; SET TRACK NUMBER
4A21H+b    JMP SETSEC     ; SET SECTOR NUMBER
4A24H+b    JMP SETDMA     ; SET DMA ADDRESS
4A27H+b    JMP READ       ; READ SELECTED SECTOR
4A2AH+b    JMP WRITE      ; WRITE SELECTED SECTOR
4A2DH+b    JMP LISTST     ; RETURN LIST STATUS
4A30H+b    JMP SECTRAN    ; SECTOR TRANSLATE SUBROUTINE
```

Each jump address corresponds to a particular subroutine which performs the specific function, as outlined below. There are three major divisions in the jump table: the system (re)initialization which results from calls on BOOT and WBOOT, simple character I/O performed by calls on CONST, CONIN, CONOUT, LIST, PUNCH, READER, and LISTST, and diskette I/O performed by calls on HOME, SELDSK, SETTRK, SETSEC, SETDMA, READ, WRITE, and SECTRAN.

All simple character I/O operations are assumed to be performed in ASCII, upper and lower case, with high order (parity bit) set to zero. An end-of-file condition for an input device is given by an ASCII control-z (1AH). Peripheral devices are seen by CP/M as "logical" devices, and are assigned to physical devices within the BIOS.

In order to operate, the BDOS needs only the CONST, CONIN, and CONOUT subroutines (LIST, PUNCH, and READER may be used by PIP, but not the BDOS). Further, the LISTST entry is used currently only by DESPOOL, and thus, the initial version of CBIOS may have empty subroutines for the remaining ASCII devices.

The characteristics of each device are

CONSOLE — The principal interactive console which communicates with the operator, accessed through CONST, CONIN, and CONOUT. Typically, the CONSOLE is a device such as a CRT or Teletype.

LIST — The principal listing device, if it exists on your system, which is usually a hard-copy device, such as a printer or Teletype.

PUNCH — The principal tape punching device, if it exists, which is normally a high-speed paper tape punch or Teletype.

READER — The principal tape reading device, such as a simple optical reader or Teletype.

Note that a single peripheral can be assigned as the LIST, PUNCH, and READER device simultaneously. If no peripheral device is assigned as the LIST, PUNCH, or READER device, the CBIOS created by the user may give an appropriate error message so that the system does not "hang" if the device is accessed by PIP or some other user program. Alternately, the PUNCH and LIST routines can just simply return, and the READER routine can return with a lAH (ctl-Z) in reg A to indicate immediate end-of-file.

For added flexibility, the user can optionally implement the "IOBYTE" function which allows reassignment of physical and logical devices. The IOBYTE function creates a mapping of logical to physical devices which can be altered during CP/M processing (see the STAT command). The definition of the IOBYTE function corresponds to the Intel standard as follows: a single location in memory (currently location ØØØ3H) is maintained, called IOBYTE, which defines the logical to physical device mapping which is in effect at a particular time. The mapping is performed by splitting the IOBYTE into four distinct fields of two bits each, called the CONSOLE, READER, PUNCH, and LIST fields, as shown below:

```
                     most significant        least significant
                     -----------------------------------------
IOBYTE AT  ØØØ3H     | LIST     | PUNCH    | READER   | CONSOLE |
                     -----------------------------------------
                      bits 6,7  bits 4,5  bits 2,3  bits Ø,1
```

The value in each field can be in the range Ø-3, defining the assigned source or destination of each logical device. The values which can be assigned to each field are given below

CONSOLE field (bits 0,1)
- 0 - console is assigned to the console printer device (TTY:)
- 1 - console is assigned to the CRT device (CRT:)
- 2 - batch mode: use the READER as the CONSOLE input, and the LIST device as the CONSOLE output (BAT:)
- 3 - user defined console device (UC1:)

READER field (bits 2,3)
- 0 - READER is the Teletype device (TTY:)
- 1 - READER is the high-speed reader device (RDR:)
- 2 - user defined reader # 1 (UR1:)
- 3 - user defined reader # 2 (UR2:)

PUNCH field (bits 4,5)
- 0 - PUNCH is the Teletype device (TTY:)
- 1 - PUNCH is the high speed punch device (PUN:)
- 2 - user defined punch # 1 (UP1:)
- 3 - user defined punch # 2 (UP2:)

LIST field (bits 6,7)
- 0 - LIST is the Teletype device (TTY:)
- 1 - LIST is the CRT device (CRT:)
- 2 - LIST is the line printer device (LPT:)
- 3 - user defined list device (UL1:)

Note again that the implementation of the IOBYTE is optional, and affects only the organization of your CBIOS. No CP/M systems use the IOBYTE (although they tolerate the existence of the IOBYTE at location 0003H), except for PIP which allows access to the physical devices, and STAT which allows logical-physical assignments to be made and/or displayed (for more information, see the "CP/M Features and Facilities Guide"). In any case, the IOBYTE implementation should be omitted until your basic CBIOS is fully implemented and tested; then add the IOBYTE to increase your facilities.

Disk I/O is always performed through a sequence of calls on the various disk access subroutines which set up the disk number to access, the track and sector on a particular disk, and the direct memory access (DMA) address involved in the I/O operation. After all these parameters have been set up, a call is made to the READ or WRITE function to perform the actual I/O operation. Note that there is often a single call to SELDSK to select a disk drive, followed by a number of read or write operations to the selected disk before selecting another drive for subsequent operations. Similarly, there may be a single call to set the DMA address, followed by several calls which read or write from the selected DMA address before the DMA address is changed. The track and sector subroutines are always called before the READ or WRITE operations are performed.

Note that the READ and WRITE routines should perform several retries (10 is standard) before reporting the error condition to the BDOS. If the error condition is returned to the BDOS, it will report the error to the user. The HOME subroutine may or may not actually perform the track 00 seek, depending upon your controller characteristics; the important point is that track 00 has been selected for the next operation, and is often treated in exactly the same manner as SETTRK with a parameter of 00.

The exact responsibilites of each entry point subroutine are given below:

BOOT — The BOOT entry point gets control from the cold start loader and is responsible for basic system initialization, including sending a signon message (which can be omitted in the first version). If the IOBYTE function is implemented, it must be set at this point. The various system parameters which are set by the WBOOT entry point must be initialized, and control is transferred to the CCP at 3400H+b for further processing. Note that reg C must be set to zero to select drive A.

WBOOT — The WBOOT entry point gets control when a warm start occurs. A warm start is performed whenever a user program branches to location 0000H, or when the CPU is reset from the front panel. The CP/M system must be loaded from the first two tracks of drive A up to, but not including, the BIOS (or CBIOS, if you have completed your patch). System parameters must be initialized as shown below:

location 0,1,2	set to JMP WBOOT for warm starts (0000H: JMP 4A03H+b)
location 3	set initial value of IOBYTE, if implemented in your CBIOS
location 5,6,7	set to JMP BDOS, which is the primary entry point to CP/M for transient programs. (0005H: JMP 3C06H+b)

(see Section 9 for complete details of page zero use) Upon completion of the initialization, the WBOOT program must branch to the CCP at 3400H+b to (re)start the system. Upon entry to the CCP, register C is set to the drive to select after system initialization.

CONST — Sample the status of the currently assigned console device and return 0FFH in register A if a character is ready to read, and 00H in register A if no console characters are ready.

CONIN — Read the next console character into register A, and set the parity bit (high order bit) to zero. If no console character is ready, wait until a character is typed before returning.

CONOUT — Send the character from register C to the console output device. The character is in ASCII, with high order parity bit set to zero. You may want to include a time-out on a line feed or carriage return, if your console device requires some time interval at the end of the line (such as a TI Silent 700 terminal). You can, if you wish, filter out control characters which cause your console device to react in a strange way (a control-z causes the Lear Seigler terminal to clear the screen, for example).

LIST — Send the character from register C to the currently assigned listing device. The character is in ASCII with zero parity.

PUNCH — Send the character from register C to the currently assigned punch device. The character is in ASCII with zero parity.

READER — Read the next character from the currently assigned reader device into register A with zero parity (high order bit must be zero), an end of file condition is reported by returning an ASCII control-z (1AH).

HOME — Return the disk head of the currently selected disk (initially disk A) to the track 00 position. If your controller allows access to the track 0 flag from the drive, step the head until the track 0 flag is detected. If your controller does not support this feature, you can translate the HOME call into a call on SETTRK with a parameter of 0.

SELDSK — Select the disk drive given by register C for further operations, where register C contains 0 for drive A, 1 for drive B, and so-forth up to 15 for drive P (the standard CP/M distribution version supports four drives). On each disk select, SELDSK must return in HL the base address of a 16-byte area, called the Disk Parameter Header, described in the Section 10. For standard floppy disk drives, the contents of the header and associated tables does not change, and thus the program segment included in the sample CBIOS performs this operation automatically. If there is an attempt to select a non-existent drive, SELDSK returns HL=0000H as an error indicator. Although SELDSK must return the header address on each call, it is advisable to postpone the actual physical disk select operation until an I/O function (seek, read or write) is actually performed, since disk selects often occur without utimately performing any disk I/O, and many controllers will unload the head of the current disk before selecting the new drive. This would cause an excessive amount of noise and disk wear.

SETTRK — Register BC contains the track number for subsequent disk accesses on the currently selected drive. You can choose to seek the selected track at this time, or delay the seek until the next read or write actually occurs. Register BC can take on values in the range 0-76 corresponding to valid track numbers for standard floppy disk drives, and 0-65535 for non-standard disk subsystems.

SETSEC — Register BC contains the sector number (1 through 26) for subsequent disk accesses on the currently selected drive. You can choose to send this information to the controller at this point, or instead delay sector selection until a read or write operation occurs.

SETDMA — Register BC contains the DMA (disk memory access) address for subsequent read or write operations. For example, if B = 00H and C = 80H when SETDMA is called, then all subsequent read operations read their data into 80H through 0FFH, and all subsequent write operations get their data from 80H through 0FFH, until the next call to SETDMA occurs. The initial DMA address is assumed to be 80H. Note that the controller need not actually support direct memory access. If, for example, all data is received and sent through I/O ports, the CBIOS which you construct will use the 128 byte area starting at the selected DMA address for the memory buffer during the following read or write operations.

READ — Assuming the drive has been selected, the track has been set, the sector has been set, and the DMA address has been specified, the READ subroutine attempts to read one sector based upon these parameters, and returns the following error codes in register A:

0	no errors occurred
1	non-recoverable error condition occurred

Currently, CP/M responds only to a zero or non-zero value as the return code. That is, if the value in register A is 0 then CP/M assumes that the disk operation completed properly. If an error occurs, however, the CBIOS should attempt at least 10 retries to see if the error is recoverable. When an error is reported the BDOS will print the message "BDOS ERR ON x: BAD SECTOR". The operator then has the option of typing <cr> to ignore the error, or ctl-C to abort.

WRITE — Write the data from the currently selected DMA address to the currently selected drive, track, and sector. The data should be marked as "non deleted data" to maintain compatibility with other CP/M systems. The error codes given in the READ command are returned in register A, with error recovery attempts as described above.

LISTST — Return the ready status of the list device. Used by the DESPOOL program to improve console response during its operation. The value 00 is returned in A if the list device is not ready to accept a character, and 0FFH if a character can be sent to the printer. Note that a 00 value always suffices.

SECTRAN — Performs sector logical to physical sector translation in order to improve the overall response of CP/M. Standard CP/M systems are shipped with a "skew factor" of 6, where six 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 which use fast processors, memory, and disk subsystems, the skew factor may be changed to improve overall response. Note, however, that you should maintain a single density IBM compatible version of CP/M for information transfer into and out of your computer system, using a skew factor of 6. In general, SECTRAN receives a logical sector number in BC, and a translate table address in DE. The sector number is used as an index into the translate table, with the resulting physical sector number in HL. For standard systems, the tables and indexing code is provided in the CBIOS and need not be changed.

7. A SAMPLE BIOS

The program shown in Appendix C can serve as a basis for your first BIOS. The simplest functions are assumed in this BIOS, so that you can enter it through the front panel, if absolutely necessary. Note that the user must alter and insert code into the subroutines for CONST, CONIN, CONOUT, READ, WRITE, and WAITIO subroutines. Storage is reserved for user-supplied code in these regions. The scratch area reserved in page zero (see Section 9) for the BIOS is used in this program, so that it could be implemented in ROM, if desired.

Once operational, this skeletal version can be enhanced to print the initial sign-on message and perform better error recovery. The subroutines for LIST, PUNCH, and READER can be filled-out, and the IOBYTE function can be implemented.

8. A SAMPLE COLD START LOADER

The program shown in Appendix D can serve as a basis for your cold start loader. The disk read function must be supplied by the user, and the program must be loaded somehow starting at location 0000. Note that space is reserved for your patch so that the total amount of storage required for the cold start loader is 128 bytes. Eventually, you will probably want to get this loader onto the first disk sector (track 0, sector 1), and cause your controller to load it into memory automatically upon system start-up. Alternatively, you may wish to place the cold start loader into ROM, and place it above the CP/M system. In this case, it will be necessary to originate the program at a higher address, and key-in a jump instruction at system start-up which branches to the loader. Subsequent warm starts will not require this key-in operation, since the entry point 'WBOOT' gets control, thus bringing the system in from disk automatically. Note also that the skeletal cold start loader has minimal error recovery, which may be enhanced on later versions.

9. RESERVED LOCATIONS IN PAGE ZERO

Main memory page zero, between locations 00H and 0FFH, contains several segments of code and data which are used during CP/M processing. The code and data areas are given below for reference purposes.

Locations from - to	Contents
0000H - 0002H	Contains a jump instruction to the warm start entry point at location 4A03H+b. This allows a simple programmed restart (JMP 0000H) or manual restart from the front panel.
0003H - 0003H	Contains the Intel standard IOBYTE, which is optionally included in the user's CBIOS, as described in Section 6.
0004H - 0004H	Current default drive number (0=A,...,15=P).
0005H - 0007H	Contains a jump instruction to the BDOS,and serves two purposes: JMP 0005H provides the primary entry point to the BDOS, as described in the manual "CP/M Interface Guide," and LHLD 0006H brings the address field of the instruction to the HL register pair. This value is the lowest address in memory used by CP/M (assuming the CCP is being overlayed). Note that the DDT program will change the address field to reflect the reduced memory size in debug mode.
0008H - 0027H	(interrupt locations 1 through 5 not used)
0030H - 0037H	(interrupt location 6, not currently used - reserved)
0038H - 003AH	Restart 7 - Contains a jump instruction into the DDT or SID program when running in debug mode for programmed breakpoints, but is not otherwise used by CP/M.
003BH - 003FH	(not currently used - reserved)
0040H - 004FH	16 byte area reserved for scratch by CBIOS, but is not used for any purpose in the distribution version of CP/M
0050H - 005BH	(not currently used - reserved)
005CH - 007CH	default file control block produced for a transient program by the Console Command Processor.
007DH - 007FH	Optional default random record position

ØØ8ØH - ØØFFH	default 128 byte disk buffer (also filled with the command line when a transient is loaded under the CCP).

Note that this information is set-up for normal operation under the CP/M system, but can be overwritten by a transient program if the BDOS facilities are not required by the transient.

If, for example, a particular program performs only simple I/O and must begin execution at location Ø, it can be first loaded into the TPA, using normal CP/M facilities, with a small memory move program which gets control when loaded (the memory move program must get control from location Ø1ØØH, which is the assumed beginning of all transient programs). The move program can then proceed to move the entire memory image down to location Ø, and pass control to the starting address of the memory load. Note that if the BIOS is overwritten, or if location Ø (containing the warm start entry point) is overwritten, then the programmer must bring the CP/M system back into memory with a cold start sequence.

10. DISK PARAMETER TABLES.

Tables are included in the BIOS which describe the particular characteristics of the disk subsystem used with CP/M. These tables can be either hand-coded, as shown in the sample CBIOS in Appendix C, or automatically generated using the DISKDEF macro library, as shown in Appendix B. The purpose here is to describe the elements of these tables.

In general, each disk drive has an associated (16-byte) disk parameter header which both contains information about the disk drive and provides a scratchpad area for certain BDOS operations. The format of the disk parameter header for each drive is shown below

Disk Parameter Header

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

where each element is a word (16-bit) value. The meaning of each Disk Parameter Header (DPH) element is

XLT — Address of the logical to physical translation vector, if used for this particular drive, or the value 0000H if no sector translation takes place (i.e, the physical and logical sector numbers are the same). Disk drives with identical sector skew factors share the same translate tables.

0000 — Scratchpad values for use within the BDOS (initial value is unimportant).

DIRBUF — Address of a 128 byte scratchpad area for directory operations within BDOS. All DPH's address the same scratchpad area.

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

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

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

Given n disk drives, the DPH's are arranged in a table whose first row of 16 bytes corresponds to drive 0, with the last row corresponding to drive n-1. The table thus appears as

```
DPBASE:
     -------------------------------------------------------------
  00 |XLT 00| 0000 | 0000 | 0000 |DIRBUF|DBP 00|CSV 00|ALV 00|
     -------------------------------------------------------------
  01 |XLT 01| 0000 | 0000 | 0000 |DIRBUF|DBP 01|CSV 01|ALV 01|
     -------------------------------------------------------------
                      (and so-forth through)
     -------------------------------------------------------------
  n-1|XLTn-1| 0000 | 0000 | 0000 |DIRBUF|DBPn-1|CSVn-1|ALVn-1|
     -------------------------------------------------------------
```

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

A responsibility of the SELDSK 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
......
SELDSK:
         ;SELECT DISK GIVEN BY BC
         LXI    H,0000H  ;ERROR CODE
         MOV    A,C      ;DRIVE OK?
         CPI    NDISKS   ;CY IF SO
         RNC             ;RET 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
```

The translation vectors (XLT 00 through XLTn-1) are located elsewhere in the BIOS, and simply correspond one-for-one with the logical sector numbers zero through the sector count-1. The Disk Parameter Block (DPB) for each drive is more complex. A particular DPB, which is addressed by one or more DPH's, takes the 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.

SPT is the total number of sectors per track

BSH is the data allocation block shift factor, determined by the data block allocation size.

EXM is the extent mask, determined by the data block allocation size and the number of disk blocks.

DSM determines the total storage capacity of the disk drive

DRM determines the total number of directory entries which can be stored on this drive ALØ,AL1 determine reserved directory blocks.

CKS is the size of the directory check vector

OFF is the number of reserved tracks at the beginning of the (logical) disk.

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

BLS	BSH	BLM
1,Ø24	3	7
2,Ø48	4	15
4,Ø96	5	31
8,192	6	63
16,384	7	127

where all values are in decimal. The value of EXM depends upon both the BLS and whether the DSM value is less than 256 or greater than 255, as shown in the following table

BLS	DSM < 256	DSM > 255
1,Ø24	Ø	N/A
2,Ø48	1	Ø
4,Ø96	3	1
8,192	7	3
16,384	15	7

The value of DSM is the maximum data block number supported by this particular drive, measured in BLS units. The product BLS times (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 the one less than the total number of directory entries, which can take on a 16-bit value. The values of ALØ and AL1, however, are determined by DRM. The two values ALØ and AL1 can together 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
```

where position 00 corresponds to the high order bit of the byte labelled AL0, and 15 corresponds to the low order bit of the byte labelled AL1. Each bit position reserves a data block for 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 until position 15). Each directory entry occupies 32 bytes, resulting in the following table

BLS	Directory Entries
1,024	32 times # bits
2,048	64 times # bits
4,096	128 times # bits
8,192	256 times # bits
16,384	512 times # bits

Thus, if DRM = 127 (128 directory entries), and BLS = 1024, then 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: if the disk drive media is removable, then CKS = (DRM+1)/4, where DRM is the last directory entry number. If the media is fixed, then set CKS = 0 (no directory records are checked in this case).

Finally, the OFF field determines the number of tracks which are skipped at the beginning of the physical disk. This value is automatically added whenever SETTRK is called, and 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, recall that several DPH's can address the same DPB if their drive characteristics are identical. Further, the DPB can be dynamically changed when a new drive is addressed by simply changing the pointer in the DPH since the BDOS copies the DPB values to a local area whenever the SELDSK function is invoked.

Returning back to the DPH for a particular drive, note that the two address values CSV and ALV remain. Both addresses reference an area of uninitialized memory following the BIOS. 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 CKS = (DRM+1)/4, then you must reserve (DRM+1)/4 bytes for directory check use. If CKS = 0, then 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 computed as (DSM/8)+1.

The CBIOS shown in Appendix C demonstrates an instance of these tables for standard 8" single density drives. It may be useful to examine this program, and compare the tabular values with the definitions given above.

11. THE DISKDEF MACRO LIBRARY.

A macro library is shown in Appendix F, called DISKDEF, which greatly simplifies the table construction process. You must have access to the MAC macro assembler, of course, to use the DISKDEF facility, while the macro library is included with all CP/M 2.0 distribution disks.

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

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

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

The remaining portion of your 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 your BIOS.

The form of the DISKDEF macro call is

```
    DISKDEF  dn,fsc,lsc,[skf],bls,dks,dir,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 allocation block size
dir     is the number of directory entries
cks     is the number of "checked" directory entries
ofs     is the track offset to logical track 00
[0]     is an optional 1.4 compatibility flag
```

The value "dn" is the drive number being defined with this DISKDEF macro invocation. The "fsc" parameter accounts for differing sector numbering systems, and is usually 0 or 1. The "lsc" is the last numbered sector on a track. 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 single-byte table is created, otherwise each translation table element occupies two bytes. No translation table is created if the skf parameter is omitted (or equal to 0). 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. Further, each directory entry addresses more data and the BIOS-resident ram space is reduced. The "dks" specifies the total disk size in "bls" units. That is, if the bls = 2048 and dks = 1000, then the total disk capacity is 2,048,000 bytes. If dks is greater than 255, then 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, and is used internally to detect changed disks during system operation, where an intervening cold or warm start has not occurred (when this situation is detected, CP/M automatically marks the disk read/only so that data is not subsequently destroyed). As stated in the previous section, the value of cks = dir when the media is easily changed, as is the case with a floppy disk subsystem. If the disk is permanently mounted, then the value of cks is typically 0, since the probability of changing disks without a restart is quite low. The "ofs" value determines the number of tracks to skip when this particular drive is addressed, which can be used to reserve additional operating system space or to simulate several logical drives on a single large capacity physical drive. Finally, the [0] parameter is included when file compatibility is required with versions of 1.4 which have been modified for higher density disks. This parameter ensures that only 16K is allocated for each directory record, as was the case for previous 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 standard four-drive single density system, which is compatible with version 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 per track (numbered 1 through 26), with 6 sectors skipped between each access, 1024 bytes per data block, 243 data blocks for a total of 243k byte disk capacity, 64 checked directory entries, and two operating system tracks.

The DISKS macro generates n Disk Parameter Headers (DPH's), starting at the DPH table address DPBASE generated by the macro. Each disk header block contains sixteen bytes, as described above, and correspond one-for-one to each of the defined drives. In the four drive standard system, for example, the DISKS macro generates a table of the form:

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

where the DPH labels are included for reference purposes to show the beginning table addresses for each drive 0 through 3. The values contained within the disk parameter header are described in detail in the previous section. The check and allocation vector addresses are generated by the ENDEF macro in the ram area following the BIOS code and tables.

Note that if the "skf" (skew factor) parameter is omitted (or equal to 0), the translation table is omitted, and a 0000H value is inserted in the XLT position of the disk parameter header for the disk. In a subsequent call to perform the logical to physical translation, SECTRAN 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 DPH's. The table shown below, for example, 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 which 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 four-drive system, the ENDEF macro might produce

```
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 use the STAT program to check your drive characteristics, since STAT uses the disk parameter block to decode the drive information. The STAT command form

STAT d:DSK:

decodes the disk parameter block for drive d (d=A,...,P) and displays the values shown below:

```
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: Sectors/ Track
t: Reserved Tracks
```

Three examples of DISKDEF macro invocations are shown below with corresponding STAT parameter values (the last produces a full 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,,2048,1024,300,0,2
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=65536, k=8192, d=128, c=128, e=1024, b=128, s=58, t=2
```

12. SECTOR BLOCKING AND DEBLOCKING.

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

Upon 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 0 occurs whenever the next write operation is into a previously written area, such as a random mode record update, 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 1 occurs when a write into the directory area is performed. Condition 2 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, and thus there is little overhead involved in either operation when blocking and deblocking records since pre-read operations can be avoided when writing records.

Appendix G lists the blocking and deblocking algorithms in skeletal form (this file is included on your CP/M disk). Generally, the algorithms map all CP/M sector read operations onto the host disk through an intermediate buffer which is the size of the host disk sector. Throughout the program, values and variables which relate to the CP/M sector involved in a seek operation are prefixed by "sek," while those related to the host disk system are prefixed by "hst." The equate statements beginning on line 29 of Appendix G define the mapping between CP/M and the host system, and must be changed if other than the sample host system is involved.

The entry points BOOT and WBOOT must contain the initialization code starting on line 57, while the SELDSK entry point must be augmented by the code starting on line 65. Note that although the SELDSK entry point computes and returns the Disk Parameter Header address, it does not physically selected the host disk at this point (it is selected later at READHST or WRITEHST). Further, SETTRK, SETTRK, and SETDMA simply store the values, but do not take any other action at this point. SECTRAN performs a trivial trivial function of returning the physical sector number.

The principal entry points are READ and WRITE, starting on lines 110 and 125, respectively. These subroutines take the place of your previous READ and WRITE operations.

The actual physical read or write takes place at either WRITEHST or READHST, where all values have been prepared: hstdsk is the host disk number, hsttrk is the host track number, and hstsec is the host sector number (which may require translation to a physical sector number). You must insert code at this point which performs the full host sector read or write into, or out of, the buffer at hstbuf of length hstsiz. All other mapping functions are performed by the algorithms.

This particular algorithm was tested using an 80 megabyte hard disk unit which was originally configured for 128 byte sectors, producing approximately 35 megabytes of formatted storage. When configured for 512 byte host sectors, usable storage increased to 57 megabytes, with a corresponding 400% improvement in overall response. In this situation, there is no apparent overhead involved in deblocking sectors, with the advantage that user programs still maintain the (less memory consuming) 128-byte sectors. This is primarily due, of course, to the information provided by the BDOS which eliminates the necessity for pre-read operations to take place.

APPENDIX A: THE MDS COLD START LOADER

```
                ;       MDS-800 Cold Start Loader for CP/M 2.0
                ;
                ;       Version 2.0 August, 1979
                ;
0000 =          false   equ     0
ffff =          true    equ     not false
0000 =          testing equ     false
                ;
                        if      testing
                bias    equ     03400h
                        endif
                        if      not testing
0000 =          bias    equ     0000h
                        endif
0000 =          cpmb    equ     bias            ;base of dos load
0806 =          bdos    equ     806h+bias       ;entry to dos for calls
1880 =          bdose   equ     1880h+bias      ;end of dos load
1600 =          boot    equ     1600h+bias      ;cold start entry point
1603 =          rboot   equ     boot+3          ;warm start entry point
                ;
3000                    org     3000h   ;loaded here by hardware
                ;
1880 =          bdosl   equ     bdose-cpmb
0002 =          ntrks   equ     2               ;tracks to read
0031 =          bdoss   equ     bdosl/128       ;# sectors in bdos
0019 =          bdos0   equ     25              ;# on track 0
0018 =          bdosl   equ     bdoss-bdos0     ;# on track 1
                ;
f800 =          mon80   equ     0f800h  ;intel monitor base
ff0f =          rmon80  equ     0ff0fh  ;restart location for mon80
0078 =          base    equ     078h    ;'base' used by controller
0079 =          rtype   equ     base+1  ;result type
007b =          rbyte   equ     base+3  ;result byte
007f =          reset   equ     base+7  ;reset controller
                ;
0078 =          dstat   equ     base    ;disk status port
0079 =          ilow    equ     base+1  ;low iopb address
007a =          ihigh   equ     base+2  ;high iopb address
00ff =          bsw     equ     0ffh    ;boot switch
0003 =          recal   equ     3h      ;recalibrate selected drive
0004 =          readf   equ     4h      ;disk read function
0100 =          stack   equ     100h    ;use end of boot for stack
                ;
                rstart:
3000 310001             lxi     sp,stack;in case of call to mon80
                ;       clear disk status
3003 db79               in      rtype
3005 db7b               in      rbyte
                ;       check if boot switch is off
                coldstart:
3007 dbff               in      bsw
3009 e602               ani     02h     ;switch on?
300b c20730             jnz     coldstart
```

```
                ;       clear the controller
300e d37f               out     reset   ;logic cleared
                ;
                ;
3010 0602               mvi     b,ntrks ;number of tracks to read
3012 214230             lxi     h,iopb0
                ;
                start:
                ;
                ;       read first/next track into cpmb
3015 7d                 mov     a,l
3016 d379               out     ilow
3018 7c                 mov     a,h
3019 d37a               out     ihigh
301b db78       wait0:  in      dstat
301d e604               ani     4
301f ca1b30             jz      wait0
                ;
                ;       check disk status
3022 db79               in      rtype
3024 e603               ani     11b
3026 fe02               cpi     2
                ;
                        if      testing
                        cnc     rmon80  ;go to monitor if 11 or 10
                        endif
                        if      not testing
3028 d20030             jnc     rstart  ;retry the load
                        endif
                ;
302b db7b               in      rbyte   ;i/o complete, check status
                ;       if not ready, then go to mon80
302d 17                 ral
302e dc0fff             cc      rmon80  ;not ready bit set
3031 1f                 rar             ;restore
3032 e61e               ani     11110b  ;overrun/addr err/seek/crc
                ;
                        if      testing
                        cnz     rmon80  ;go to monitor
                        endif
                        if      not testing
3034 c20030             jnz     rstart  ;retry the load
                        endif
                ;
                ;
3037 110700             lxi     d,iopbl ;length of iopb
303a 19                 dad     d       ;addressing next iopb
303b 05                 dcr     b       ;count down tracks
303c c21530             jnz     start
                ;
                ;
                ;       jmp boot, print message, set-up jmps
303f c30016             jmp     boot
                ;
                ;       parameter blocks
```

```
3042 80       iopb0:   db      80h      ;iocw, no update
3043 04                db      readf    ;read function
3044 19                db      bdos0    ;# sectors to read trk 0
3045 00                db      0        ;track 0
3046 02                db      2        ;start with sector 2, trk 0
3047 0000              dw      cpmb     ;start at base of bdos
0007 =        iopbl    equ     $-iopb0
              ;
3049 80       iopb1:   db      80h
304a 04                db      readf
304b 18                db      bdos1    ;sectors to read on track 1
304c 01                db      1        ;track 1
304d 01                db      1        ;sector 1
304e 800c              dw      cpmb+bdos0*128  ;base of second rd
3050                   end
```

APPENDIX B: THE MDS BASIC I/O SYSTEM (BIOS)

```
                ;       mds-8ØØ i/o drivers for cp/m 2.Ø
                ;       (four drive single density version)
                ;
                ;       version 2.Ø august, 1979
                ;
ØØ14 =          vers    equ     2Ø      ;version 2.Ø
                ;
                ;       copyright (c) 1979
                ;       digital research
                ;       box 579, pacific grove
                ;       california, 9395Ø
                ;
4aØØ                    org     4aØØh   ;base of bios in 2Øk system
34ØØ =          cpmb    equ     34ØØh   ;base of cpm ccp
3cØ6 =          bdos    equ     3cØ6h   ;base of bdos in 2Øk system
16ØØ =          cpml    equ     $-cpmb  ;length (in bytes) of cpm system
ØØ2c =          nsects  equ     cpml/128;number of sectors to load
ØØØ2 =          offset  equ     2       ;number of disk tracks used by cp
ØØØ4 =          cdisk   equ     ØØØ4h   ;address of last logged disk
ØØ8Ø =          buff    equ     ØØ8Øh   ;default buffer address
ØØØa =          retry   equ     1Ø      ;max retries on disk i/o before e
                ;
                ;       perform following functions
                ;       boot    cold start
                ;       wboot   warm start (save i/o byte)
                ;       (boot and wboot are the same for mds)
                ;       const   console status
                ;               reg-a = ØØ if no character ready
                ;               reg-a = ff if character ready
                ;       conin   console character in (result in reg-a)
                ;       conout  console character out (char in reg-c)
                ;       list    list out (char in reg-c)
                ;       punch   punch out (char in reg-c)
                ;       reader  paper tape reader in (result to reg-a)
                ;       home    move to track ØØ
                ;
                ;       (the following calls set-up the io parameter bloc
                ;       mds, which is used to perform subsequent reads an
                ;       seldsk  select disk given by reg-c (Ø,1,2...)
                ;       settrk  set track address (Ø,...76) for sub r/w
                ;       setsec  set sector address (1,...,26)
                ;       setdma  set subsequent dma address (initially 8Øh
                ;
                ;       read/write assume previous calls to set i/o parms
                ;       read    read track/sector to preset dma address
                ;       write   write track/sector from preset dma addres
                ;
                ;       jump vector for indiviual routines
4aØØ c3b34a             jmp     boot
4aØ3 c3c34a     wboote: jmp     wboot
4aØ6 c3614b             jmp     const
4aØ9 c3644b             jmp     conin
4aØc c36a4b             jmp     conout
```

```
4a0f c36d4b          jmp     list
4a12 c3724b          jmp     punch
4a15 c3754b          jmp     reader
4a18 c3784b          jmp     home
4a1b c37d4b          jmp     seldsk
4a1e c3a74b          jmp     settrk
4a21 c3ac4b          jmp     setsec
4a24 c3bb4b          jmp     setdma
4a27 c3c14b          jmp     read
4a2a c3ca4b          jmp     write
4a2d c3704b          jmp     listst  ;list status
4a30 c3b14b          jmp     sectran
            ;
                     maclib  diskdef ;load the disk definition library
                     disks   4       ;four disks
4a33+=      dpbase   equ     $       ;base of disk parameter blocks
4a33+824a00 dpe0:    dw      xlt0,0000h      ;translate table
4a37+000000          dw      0000h,0000h     ;scratch area
4a3b+6e4c73          dw      dirbuf,dpb0     ;dir buff,parm block
4a3f+0d4dee          dw      csv0,alv0       ;check, alloc vectors
4a43+824a00 dpe1:    dw      xlt1,0000h      ;translate table
4a47+000000          dw      0000h,0000h     ;scratch area
4a4b+6e4c73          dw      dirbuf,dpb1     ;dir buff,parm block
4a4f+3c4d1d          dw      csv1,alv1       ;check, alloc vectors
4a53+824a00 dpe2:    dw      xlt2,0000h      ;translate table
4a57+000000          dw      0000h,0000h     ;scratch area
4a5b+6e4c73          dw      dirbuf,dpb2     ;dir buff,parm block
4a5f+6b4d4c          dw      csv2,alv2       ;check, alloc vectors
4a63+824a00 dpe3:    dw      xlt3,0000h      ;translate table
4a67+000000          dw      0000h,0000h     ;scratch area
4a6b+6e4c73          dw      dirbuf,dpb3     ;dir buff,parm block
4a6f+9a4d7b          dw      csv3,alv3       ;check, alloc vectors
                     diskdef 0,1,26,6,1024,243,64,64,offset
4a73+=      dpb0     equ     $               ;disk parm block
4a73+1a00            dw      26              ;sec per track
4a75+03              db      3               ;block shift
4a76+07              db      7               ;block mask
4a77+00              db      0               ;extnt mask
4a78+f200            dw      242             ;disk size-1
4a7a+3f00            dw      63              ;directory max
4a7c+c0              db      192             ;alloc0
4a7d+00              db      0               ;alloc1
4a7e+1000            dw      16              ;check size
4a80+0200            dw      2               ;offset
4a82+=      xlt0     equ     $               ;translate table
4a82+01              db      1
4a83+07              db      7
4a84+0d              db      13
4a85+13              db      19
4a86+19              db      25
4a87+05              db      5
4a88+0b              db      11
4a89+11              db      17
4a8a+17              db      23
4a8b+03              db      3
```

```
4a8c+09             db      9
4a8d+0f             db      15
4a8e+15             db      21
4a8f+02             db      2
4a90+08             db      8
4a91+0e             db      14
4a92+14             db      20
4a93+1a             db      26
4a94+06             db      6
4a95+0c             db      12
4a96+12             db      18
4a97+18             db      24
4a98+04             db      4
4a99+0a             db      10
4a9a+10             db      16
4a9b+16             db      22
                    diskdef 1,0
4a73+=      dpb1    equ     dpb0    ;equivalent parameters
001f+=      als1    equ     als0    ;same allocation vector size
0010+=      css1    equ     css0    ;same checksum vector size
4a82+=      xlt1    equ     xlt0    ;same translate table
                    diskdef 2,0
4a73+=      dpb2    equ     dpb0    ;equivalent parameters
001f+=      als2    equ     als0    ;same allocation vector size
0010+=      css2    equ     css0    ;same checksum vector size
4a82+=      xlt2    equ     xlt0    ;same translate table
                    diskdef 3,0
4a73+=      dpb3    equ     dpb0    ;equivalent parameters
001f+=      als3    equ     als0    ;same allocation vector size
0010+=      css3    equ     css0    ;same checksum vector size
4a82+=      xlt3    equ     xlt0    ;same translate table
            ;       endef occurs at end of assembly
            ;
            ;       end of controller - independent code, the remaini
            ;       are tailored to the particular operating environm
            ;       be altered for any system which differs from the
            ;
            ;       the following code assumes the mds monitor exists
            ;       and uses the i/o subroutines within the monitor
            ;
            ;       we also assume the mds system has four disk drive
00fd =      revrt   equ     0fdh    ;interrupt revert port
00fc =      intc    equ     0fch    ;interrupt mask port
00f3 =      icon    equ     0f3h    ;interrupt control port
007e =      inte    equ     0111$1110b;enable rst 0(warm boot),rst 7
            ;
            ;       mds monitor equates
f800 =      mon80   equ     0f800h  ;mds monitor
ff0f =      rmon80  equ     0ff0fh  ;restart mon80 (boot error)
f803 =      ci      equ     0f803h  ;console character to reg-a
f806 =      ri      equ     0f806h  ;reader in to reg-a
f809 =      co      equ     0f809h  ;console char from c to console o
f80c =      po      equ     0f80ch  ;punch char from c to punch devic
f80f =      lo      equ     0f80fh  ;list from c to list device
f812 =      csts    equ     0f812h  ;console status 00/ff to register
```

```
                ;
                ;       disk ports and commands
0078 =          base    equ     78h     ;base of disk command io ports
0078 =          dstat   equ     base    ;disk status (input)
0079 =          rtype   equ     base+1  ;result type (input)
007b =          rbyte   equ     base+3  ;result byte (input)
                ;
0079 =          ilow    equ     base+1  ;iopb low address (output)
007a =          ihigh   equ     base+2  ;iopb high address (output)
                ;
0004 =          readf   equ     4h      ;read function
0006 =          writf   equ     6h      ;write function
0003 =          recal   equ     3h      ;recalibrate drive
0004 =          iordy   equ     4h      ;i/o finished mask
000d =          cr      equ     0dh     ;carriage return
000a =          lf      equ     0ah     ;line feed
                ;
                signon: ;signon message: xxk cp/m vers y.y
4a9c 0d0a0a             db      cr,lf,lf
4a9f 3230               db      '20'    ;sample memory size
4aa1 6b2043f            db      'k cp/m vers '
4aad 322e30             db      vers/10+'0','.',vers mod 10+'0'
4ab0 0d0a00             db      cr,lf,0
                ;
                boot:   ;print signon message and go to ccp
                ;       (note: mds boot initialized iobyte at 0003h)
4ab3 310001             lxi     sp,buff+80h
4ab6 219c4a             lxi     h,signon
4ab9 cdd34b             call    prmsg   ;print message
4abc af                 xra     a       ;clear accumulator
4abd 320400             sta     cdisk   ;set initially to disk a
4ac0 c30f4b             jmp     gocpm   ;go to cp/m
                ;
                ;
                wboot:; loader on track 0, sector 1, which will be skippe
                ;       read cp/m from disk - assuming there is a 128 byt
                ;       start.
                ;
4ac3 318000             lxi     sp,buff ;using dma - thus 80 thru ff ok f
                ;
4ac6 0e0a               mvi     c,retry ;max retries
4ac8 c5                 push    b
                wboot0: ;enter here on error retries
4ac9 010034             lxi     b,cpmb  ;set dma address to start of disk
4acc cdbb4b             call    setdma
4acf 0e00               mvi     c,0     ;boot from drive 0
4ad1 cd7d4b             call    seldsk
4ad4 0e00               mvi     c,0
4ad6 cda74b             call    settrk  ;start with track 0
4ad9 0e02               mvi     c,2     ;start reading sector 2
4adb cdac4b             call    setsec
                ;
                ;       read sectors, count nsects to zero
4ade c1                 pop     b       ;10-error count
4adf 062c               mvi     b,nsects
```

```
            rdsec:  ;read next sector
4ae1 c5             push    b       ;save sector count
4ae2 cdc14b         call    read
4ae5 c2494b         jnz     booterr ;retry if errors occur
4ae8 2a6c4c         lhld    iod     ;increment dma address
4aeb 118000         lxi     d,128   ;sector size
4aee 19             dad     d       ;incremented dma address in hl
4aef 44             mov     b,h
4af0 4d             mov     c,l     ;ready for call to set dma
4af1 cdbb4b         call    setdma
4af4 3a6b4c         lda     ios     ;sector number just read
4af7 fe1a           cpi     26      ;read last sector?
4af9 da054b         jc      rd1
            ;       must be sector 26, zero and go to next track
4afc 3a6a4c         lda     iot     ;get track to register a
4aff 3c             inr     a
4b00 4f             mov     c,a     ;ready for call
4b01 cda74b         call    settrk
4b04 af             xra     a       ;clear sector number
4b05 3c     rd1:    inr     a       ;to next sector
4b06 4f             mov     c,a     ;ready for call
4b07 cdac4b         call    setsec
4b0a c1             pop     b       ;recall sector count
4b0b 05             dcr     b       ;done?
4b0c c2e14a         jnz     rdsec
            ;
            ;       done with the load, reset default buffer address
            gocpm:  ;(enter here from cold start boot)
            ;       enable rst0 and rst7
4b0f f3             di
4b10 3e12           mvi     a,12h   ;initialize command
4b12 d3fd           out     revrt
4b14 af             xra     a
4b15 d3fc           out     intc    ;cleared
4b17 3e7e           mvi     a,inte  ;rst0 and rst7 bits on
4b19 d3fc           out     intc
4b1b af             xra     a
4b1c d3f3           out     icon    ;interrupt control
            ;
            ;       set default buffer address to 80h
4b1e 018000         lxi     b,buff
4b21 cdbb4b         call    setdma
            ;
            ;       reset monitor entry points
4b24 3ec3           mvi     a,jmp
4b26 320000         sta     0
4b29 21034a         lxi     h,wboote
4b2c 220100         shld    1       ;jmp wboot at location 00
4b2f 320500         sta     5
4b32 21063c         lxi     h,bdos
4b35 220600         shld    6       ;jmp bdos at location 5
4b38 323800         sta     7*8     ;jmp to mon80 (may have been chan
4b3b 2100f8         lxi     h,mon80
4b3e 223900         shld    7*8+1
            ;       leave iobyte set
```

```
                ;       previously selected disk was b, send parameter to
4b41 3a0400             lda     cdisk   ;last logged disk number
4b44 4f                 mov     c,a     ;send to ccp to log it in
4b45 fb                 ei
4b46 c30034             jmp     cpmb
                ;
                ;       error condition occurred, print message and retry
                booterr:
4b49 cl                 pop     b       ;recall counts
4b4a 0d                 dcr     c
4b4b ca524b             jz      booter0
                ;       try again
4b4e c5                 push    b
4b4f c3c94a             jmp     wboot0
                ;
                booter0:
                ;       otherwise too many retries
4b52 215b4b             lxi     h,bootmsg
4b55 cdd34b             call    prmsg
4b58 c30fff             jmp     rmon80  ;mds hardware monitor
                ;
                bootmsg:
4b5b 3f626f4            db      '?boot',0
                ;
                ;
                const:  ;console status to reg-a
                ;       (exactly the same as mds call)
4b61 c312f8             jmp     csts
                ;
                conin:  ;console character to reg-a
4b64 cd03f8             call    ci
4b67 e67f               ani     7fh     ;remove parity bit
4b69 c9                 ret
                ;
                conout: ;console character from c to console out
4b6a c309f8             jmp     co
                ;
                list:   ;list device out
                ;       (exactly the same as mds call)
4b6d c30ff8             jmp     lo
                ;
                listst:
                        ;return list status
4b70 af                 xra     a
4b71 c9                 ret             ;always not ready
                ;
                punch:  ;punch device out
                ;       (exactly the same as mds call)
4b72 c30cf8             jmp     po
                ;
                reader: ;reader character in to reg-a
                ;       (exactly the same as mds call)
4b75 c306f8             jmp     ri
                ;
                home:   ;move to home position
```

```
                ;       treat as track 00 seek
4b78 0e00               mvi     c,0
4b7a c3a74b             jmp     settrk
                ;
                seldsk: ;select disk given by register c
4b7d 210000             lxi     h,0000h ;return 0000 if error
4b80 79                 mov     a,c
4b81 fe04               cpi     ndisks  ;too large?
4b83 d0                 rnc             ;leave hl = 0000
                ;
4b84 e602               ani     10b     ;00 00 for drive 0,1 and 10 10 fo
4b86 32664c             sta     dbank   ;to select drive bank
4b89 79                 mov     a,c     ;00, 01, 10, 11
4b8a e601               ani     1b      ;mds has 0,1 at 78, 2,3 at 88
4b8c b7                 ora     a       ;result 00?
4b8d ca924b             jz      setdrive
4b90 3e30               mvi     a,00110000b     ;selects drive 1 in bank
                setdrive:
4b92 47                 mov     b,a     ;save the function
4b93 21684c             lxi     h,iof   ;io function
4b96 7e                 mov     a,m
4b97 e6cf               ani     11001111b       ;mask out disk number
4b99 b0                 ora     b       ;mask in new disk number
4b9a 77                 mov     m,a     ;save it in iopb
4b9b 69                 mov     l,c
4b9c 2600               mvi     h,0     ;hl=disk number
4b9e 29                 dad     h       ;*2
4b9f 29                 dad     h       ;*4
4ba0 29                 dad     h       ;*8
4ba1 29                 dad     h       ;*16
4ba2 11334a             lxi     d,dpbase
4ba5 19                 dad     d       ;hl=disk header table address
4ba6 c9                 ret
                ;
                ;
                settrk: ;set track address given by c
4ba7 216a4c             lxi     h,iot
4baa 71                 mov     m,c
4bab c9                 ret
                ;
                setsec: ;set sector number given by c
4bac 216b4c             lxi     h,ios
4baf 71                 mov     m,c
4bb0 c9                 ret
                sectran:
                                ;translate sector bc using table at de
4bb1 0600               mvi     b,0     ;double precision sector number i
4bb3 eb                 xchg            ;translate table address to hl
4bb4 09                 dad     b       ;translate(sector) address
4bb5 7e                 mov     a,m     ;translated sector number to a
4bb6 326b4c             sta     ios
4bb9 6f                 mov     l,a     ;return sector number in l
4bba c9                 ret
                ;
                setdma: ;set dma address given by regs b,c
```

```
4bbb 69             mov     l,c
4bbc 60             mov     h,b
4bbd 226c4c         shld    iod
4bc0 c9             ret
              ;
              read:   ;read next disk record (assuming disk/trk/sec/dma
4bc1 0e04           mvi     c,readf ;set to read function
4bc3 cde04b         call    setfunc
4bc6 cdf04b         call    waitio  ;perform read function
4bc9 c9             ret             ;may have error set in reg-a
              ;
              ;
              write:  ;disk write function
4bca 0e06           mvi     c,writf
4bcc cde04b         call    setfunc ;set to write function
4bcf cdf04b         call    waitio
4bd2 c9             ret             ;may have error set
              ;
              ;
              ;       utility subroutines
              prmsg:  ;print message at h,l to 0
4bd3 7e             mov     a,m
4bd4 b7             ora     a       ;zero?
4bd5 c8             rz
              ;       more to print
4bd6 e5             push    h
4bd7 4f             mov     c,a
4bd8 cd6a4b         call    conout
4bdb e1             pop     h
4bdc 23             inx     h
4bdd c3d34b         jmp     prmsg
              ;
              setfunc:
              ;       set function for next i/o (command in reg-c)
4be0 21684c         lxi     h,iof   ;io function address
4be3 7e             mov     a,m     ;get it to accumulator for maskin
4be4 e6f8           ani     11111000b       ;remove previous command
4be6 b1             ora     c       ;set to new command
4be7 77             mov     m,a     ;replaced in iopb
              ;       the mds-800 controller req's disk bank bit in sec
              ;       mask the bit from the current i/o function
4be8 e620           ani     00100000b       ;mask the disk select bit
4bea 216b4c         lxi     h,ios           ;address the sector selec
4bed b6             ora     m               ;select proper disk bank
4bee 77             mov     m,a             ;set disk select bit on/o
4bef c9             ret
              ;
              waitio:
4bf0 0e0a           mvi     c,retry ;max retries before perm error
              rewait:
              ;       start the i/o function and wait for completion
4bf2 cd3f4c         call    intype  ;in rtype
4bf5 cd4c4c         call    inbyte  ;clears the controller
              ;
4bf8 3a664c         lda     dbank           ;set bank flags
```

```
4bfb b7                 ora     a                 ;zero if drive 0,1 and nz
4bfc 3e67               mvi     a,iopb and 0ffh ;low address for iopb
4bfe 064c               mvi     b,iopb shr 8    ;high address for iopb
4c00 c20b4c             jnz     iodrl   ;drive bank 1?
4c03 d379               out     ilow              ;low address to controlle
4c05 78                 mov     a,b
4c06 d37a               out     ihigh   ;high address
4c08 c3104c             jmp     wait0             ;to wait for complete
            ;
            iodrl:  ;drive bank 1
4c0b d389               out     ilow+10h          ;88 for drive bank 10
4c0d 78                 mov     a,b
4c0e d38a               out     ihigh+10h
            ;
4c10 cd594c wait0:  call    instat            ;wait for completion
4c13 e604               ani     iordy             ;ready?
4c15 ca104c             jz      wait0
            ;
            ;       check io completion ok
4c18 cd3f4c             call    intype            ;must be io complete (00)
            ;       00 unlinked i/o complete,    01 linked i/o comple
            ;       10 disk status changed       11 (not used)
4c1b fe02               cpi     10b               ;ready status change?
4c1d ca324c             jz      wready
            ;
            ;       must be 00 in the accumulator
4c20 b7                 ora     a
4c21 c2384c             jnz     werror            ;some other condition, re
            ;
            ;       check i/o error bits
4c24 cd4c4c             call    inbyte
4c27 17                 ral
4c28 da324c             jc      wready            ;unit not ready
4c2b 1f                 rar
4c2c e6fe               ani     11111110b         ;any other errors?
4c2e c2384c             jnz     werror
            ;
            ;       read or write is ok, accumulator contains zero
4c31 c9                 ret
            ;
            wready: ;not ready, treat as error for now
4c32 cd4c4c             call    inbyte            ;clear result byte
4c35 c3384c             jmp     trycount
            ;
            werror: ;return hardware malfunction (crc, track, seek, e
            ;       the mds controller has returned a bit in each pos
            ;       of the accumulator, corresponding to the conditio
            ;       0       - deleted data (accepted as ok above)
            ;       1       - crc error
            ;       2       - seek error
            ;       3       - address error (hardware malfunction)
            ;       4       - data over/under flow (hardware malfunct
            ;       5       - write protect (treated as not ready)
            ;       6       - write error (hardware malfunction)
            ;       7       - not ready
```

```
            ;       (accumulator bits are numbered 7 6 5 4 3 2 1 0)
            ;
            ;       it may be useful to filter out the various condit
            ;       but we will get a permanent error message if it i
            ;       recoverable. in any case, the not ready conditio
            ;       treated as a separate condition for later improve
            trycount:
            ;       register c contains retry count, decrement 'til z
4c38 0d             dcr     c
4c39 c2f24b         jnz     rewait  ;for another try
            ;
            ;       cannot recover from error
4c3c 3e01           mvi     a,1     ;error code
4c3e c9             ret
            ;
            ;       intype, inbyte, instat read drive bank 00 or 10
4c3f 3a664c intype: lda     dbank
4c42 b7             ora     a
4c43 c2494c         jnz     intypl  ;skip to bank 10
4c46 db79           in      rtype
4c48 c9             ret
4c49 db89   intypl: in      rtype+10h       ;78 for 0,1  88 for 2,3
4c4b c9             ret
            ;
4c4c 3a664c inbyte: lda     dbank
4c4f b7             ora     a
4c50 c2564c         jnz     inbytl
4c53 db7b           in      rbyte
4c55 c9             ret
4c56 db8b   inbytl: in      rbyte+10h
4c58 c9             ret
            ;
4c59 3a664c instat: lda     dbank
4c5c b7             ora     a
4c5d c2634c         jnz     instal
4c60 db78           in      dstat
4c62 c9             ret
4c63 db88   instal: in      dstat+10h
4c65 c9             ret
            ;
            ;
            ;
            ;       data areas (must be in ram)
4c66 00     dbank:  db      0       ;disk bank 00 if drive 0,1
                                    ;          10 if drive 2,3
            iopb:   ;io parameter block
4c67 80             db      80h     ;normal i/o operation
4c68 04     iof:    db      readf   ;io function, initial read
4c69 01     ion:    db      1       ;number of sectors to read
4c6a 02     iot:    db      offset  ;track number
4c6b 01     ios:    db      1       ;sector number
4c6c 8000   iod:    dw      buff    ;io address
            ;
            ;
            ;       define ram areas for bdos operation
```

```
                        endef
4c6e+=      begdat   equ     $
4c6e+       dirbuf:  ds      128     ;directory access buffer
4cee+       alvØ:    ds      31
4dØd+       csvØ:    ds      16
4d1d+       alv1:    ds      31
4d3c+       csv1:    ds      16
4d4c+       alv2:    ds      31
4d6b+       csv2:    ds      16
4d7b+       alv3:    ds      31
4d9a+       csv3:    ds      16
4daa+=      enddat   equ     $
Ø13c+=      datsiz   equ     $-begdat
4daa                 end
```

```
                ;       skeletal cbios for first level of cp/m 2.0 altera
                ;
0014 =          msize   equ     20      ;cp/m version memory size in kilo
                ;
                ;       "bias" is address offset from 3400h for memory sy
                ;       than 16k (referred to as "b" throughout the text)
                ;
0000 =          bias    equ     (msize-20)*1024
3400 =          ccp     equ     3400h+bias      ;base of ccp
3c06 =          bdos    equ     ccp+806h        ;base of bdos
4a00 =          bios    equ     ccp+1600h       ;base of bios
0004 =          cdisk   equ     0004h   ;current disk number 0=a,...,15=p
0003 =          iobyte  equ     0003h   ;intel i/o byte
                ;
4a00                    org     bios    ;origin of this program
002c =          nsects  equ     ($-ccp)/128     ;warm start sector count
                ;
                ;       jump vector for individual subroutines
4a00 c39c4a             jmp     boot            ;cold start
4a03 c3a64a     wboote: jmp     wboot           ;warm start
4a06 c3114b             jmp     const           ;console status
4a09 c3244b             jmp     conin           ;console character in
4a0c c3374b             jmp     conout          ;console character out
4a0f c3494b             jmp     list            ;list character out
4a12 c34d4b             jmp     punch           ;punch character out
4a15 c34f4b             jmp     reader          ;reader character out
4a18 c3544b             jmp     home            ;move head to home positi
4a1b c35a4b             jmp     seldsk          ;select disk
4a1e c37d4b             jmp     settrk          ;set track number
4a21 c3924b             jmp     setsec          ;set sector number
4a24 c3ad4b             jmp     setdma          ;set dma address
4a27 c3c34b             jmp     read            ;read disk
4a2a c3d64b             jmp     write           ;write disk
4a2d c34b4b             jmp     listst          ;return list status
4a30 c3a74b             jmp     sectran         ;sector translate
                ;
                ;       fixed data tables for four-drive standard
                ;       ibm-compatible 8" disks
                ;       disk parameter header for disk 00
4a33 734a00     dpbase: dw      trans,0000h
4a37 000000             dw      0000h,0000h
4a3b f04c8d             dw      dirbf,dpblk
4a3f ec4d70             dw      chk00,all00
                ;       disk parameter header for disk 01
4a43 734a00             dw      trans,0000h
4a47 000000             dw      0000h,0000h
4a4b f04c8d             dw      dirbf,dpblk
4a4f fc4d8f             dw      chk01,all01
                ;       disk parameter header for disk 02
4a53 734a00             dw      trans,0000h
4a57 000000             dw      0000h,0000h
4a5b f04c8d             dw      dirbf,dpblk
4a5f 0c4eae             dw      chk02,all02
```

```
                ;       disk parameter header for disk 03
4a63 734a00             dw      trans,0000h
4a67 000000             dw      0000h,0000h
4a6b f04c8d             dw      dirbf,dpblk
4a6f 1c4ecd             dw      chk03,all03
                ;
                ;       sector translate vector
4a73 01070d     trans:  db      1,7,13,19               ;sectors 1,2,3,4
4a77 19050b             db      25,5,11,17              ;sectors 5,6,7,8
4a7b 170309             db      23,3,9,15               ;sectors 9,10,11,12
4a7f 150208             db      21,2,8,14               ;sectors 13,14,15,16
4a83 141a06             db      20,26,6,12              ;sectors 17,18,19,20
4a87 121804             db      18,24,4,10              ;sectors 21,22,23,24
4a8b 1016               db      16,22                   ;sectors 25,26
                ;
                dpblk:  ;disk parameter block, common to all disks
4a8d 1a00               dw      26                      ;sectors per track
4a8f 03                 db      3                       ;block shift factor
4a90 07                 db      7                       ;block mask
4a91 00                 db      0                       ;null mask
4a92 f200               dw      242                     ;disk size-1
4a94 3f00               dw      63                      ;directory max
4a96 c0                 db      192                     ;alloc 0
4a97 00                 db      0                       ;alloc 1
4a98 1000               dw      16                      ;check size
4a9a 0200               dw      2                       ;track offset
                ;
                ;       end of fixed tables
                ;
                ;       individual subroutines to perform each function
                boot:   ;simplest case is to just perform parameter initi
4a9c af                 xra     a                       ;zero in the accum
4a9d 320300             sta     iobyte                  ;clear the iobyte
4aa0 320400             sta     cdisk                   ;select disk zero
4aa3 c3ef4a             jmp     gocpm                   ;initialize and go to cp/
                ;
                wboot:  ;simplest case is to read the disk until all sect
4aa6 318000             lxi     sp,80h                  ;use space below buffer f
4aa9 0e00               mvi     c,0                     ;select disk 0
4aab cd5a4b             call    seldsk
4aae cd544b             call    home                    ;go to track 00
                ;
4ab1 062c               mvi     b,nsects                ;b counts # of sectors to
4ab3 0e00               mvi     c,0                     ;c has the current track
4ab5 1602               mvi     d,2                     ;d has the next sector to
                ;       note that we begin by reading track 0, sector 2 s
                ;       contains the cold start loader, which is skipped
4ab7 210034             lxi     h,ccp                   ;base of cp/m (initial lo
                load1:  ;load one more sector
4aba c5                 push    b       ;save sector count, current track
4abb d5                 push    d       ;save next sector to read
4abc e5                 push    h       ;save dma address
4abd 4a                 mov     c,d     ;get sector address to register c
4abe cd924b             call    setsec  ;set sector address from register
4ac1 c1                 pop     b       ;recall dma address to b,c
```

```
4ac2 c5                 push    b       ;replace on stack for later recal
4ac3 cdad4b             call    setdma  ;set dma address from b,c
            ;
            ;           drive set to Ø, track set, sector set, dma addres
4ac6 cdc34b             call    read
4ac9 feØØ               cpi     ØØh     ;any errors?
4acb c2a64a             jnz     wboot   ;retry the entire boot if an erro
            ;
            ;           no error, move to next sector
4ace e1                 pop     h       ;recall dma address
4acf 118ØØØ             lxi     d,128   ;dma=dma+128
4ad2 19                 dad     d       ;new dma address is in h,l
4ad3 d1                 pop     d       ;recall sector address
4ad4 c1                 pop     b       ;recall number of sectors remaini
4ad5 Ø5                 dcr     b       ;sectors=sectors-1
4ad6 caef4a             jz      gocpm   ;transfer to cp/m if all have bee
            ;
            ;           more sectors remain to load, check for track chan
4ad9 14                 inr     d
4ada 7a                 mov     a,d     ;sector=27?, if so, change tracks
4adb fe1b               cpi     27
4add daba4a             jc      load1   ;carry generated if sector<27
            ;
            ;           end of current track, go to next track
4aeØ 16Ø1               mvi     d,1     ;begin with first sector of next
4ae2 Øc                 inr     c       ;track=track+1
            ;
            ;           save register state, and change tracks
4ae3 c5                 push    b
4ae4 d5                 push    d
4ae5 e5                 push    h
4ae6 cd7d4b             call    settrk  ;track address set from register
4ae9 e1                 pop     h
4aea d1                 pop     d
4aeb c1                 pop     b
4aec c3ba4a             jmp     load1   ;for another sector
            ;
            ;           end of load operation, set parameters and go to c
            gocpm:
4aef 3ec3               mvi     a,Øc3h  ;c3 is a jmp instruction
4af1 32ØØØØ             sta     Ø       ;for jmp to wboot
4af4 21Ø34a             lxi     h,wboote        ;wboot entry point
4af7 22Ø1ØØ             shld    1       ;set address field for jmp at Ø
            ;
4afa 32Ø5ØØ             sta     5       ;for jmp to bdos
4afd 21Ø63c             lxi     h,bdos  ;bdos entry point
4bØØ 22Ø6ØØ             shld    6       ;address field of jump at 5 to bd
            ;
4bØ3 Ø18ØØØ             lxi     b,8Øh   ;default dma address is 8Øh
4bØ6 cdad4b             call    setdma
            ;
4bØ9 fb                 ei              ;enable the interrupt system
4bØa 3aØ4ØØ             lda     cdisk   ;get current disk number
4bØd 4f                 mov     c,a     ;send to the ccp
4bØe c3ØØ34             jmp     ccp     ;go to cp/m for further processin
```

```
            ;
            ;
            ;       simple i/o handlers (must be filled in by user)
            ;       in each case, the entry point is provided, with s
            ;       to insert your own code
            ;
            const:  ;console status, return 0ffh if character ready,
4b11                ds      10h     ;space for status subroutine
4b21 3e00           mvi     a,00h
4b23 c9             ret
            ;
            conin:  ;console character into register a
4b24                ds      10h     ;space for input routine
4b34 e67f           ani     7fh     ;strip parity bit
4b36 c9             ret
            ;
            conout: ;console character output from register c
4b37 79             mov     a,c     ;get to accumulator
4b38                ds      10h     ;space for output routine
4b48 c9             ret
            ;
            list:   ;list character from register c
4b49 79             mov     a,c     ;character to register a
4b4a c9             ret             ;null subroutine
            ;
            listst: ;return list status (0 if not ready, 1 if ready)
4b4b af             xra     a       ;0 is always ok to return
4b4c c9             ret
            ;
            punch:  ;punch character from register c
4b4d 79             mov     a,c     ;character to register a
4b4e c9             ret             ;null subroutine
            ;
            ;
            reader: ;read character into register a from reader devic
4b4f 3e1a           mvi     a,1ah   ;enter end of file for now (repla
4b51 e67f           ani     7fh     ;remember to strip parity bit
4b53 c9             ret
            ;
            ;
            ;       i/o drivers for the disk follow
            ;       for now, we will simply store the parameters away
            ;       in the read and write subroutines
            ;
            home:   ;move to the track 00 position of current drive
            ;       translate this call into a settrk call with param
4b54 0e00           mvi     c,0     ;select track 0
4b56 cd7d4b         call    settrk
4b59 c9             ret             ;we will move to 00 on first read
            ;
            seldsk: ;select disk given by register c
4b5a 210000         lxi     h,0000h ;error return code
4b5d 79             mov     a,c
4b5e 32ef4c         sta     diskno
4b61 fe04           cpi     4       ;must be between 0 and 3
```

```
4b63 dØ             rnc                ;no carry if 4,5,...
               ;    disk number is in the proper range
4b64                ds      1Ø         ;space for disk select
               ;    compute proper disk parameter header address
4b6e 3aef4c         lda     diskno
4b71 6f             mov     l,a        ;l=disk number Ø,1,2,3
4b72 26ØØ           mvi     h,Ø        ;high order zero
4b74 29             dad     h          ;*2
4b75 29             dad     h          ;*4
4b76 29             dad     h          ;*8
4b77 29             dad     h          ;*16 (size of each header)
4b78 11334a         lxi     d,dpbase
4b7b 19             dad     d          ;hl=.dpbase(diskno*16)
4b7c c9             ret
               ;
               settrk: ;set track given by register c
4b7d 79             mov     a,c
4b7e 32e94c         sta     track
4b81                ds      1Øh        ;space for track select
4b91 c9             ret
               ;
               setsec: ;set sector given by register c
4b92 79             mov     a,c
4b93 32eb4c         sta     sector
4b96                ds      1Øh        ;space for sector select
4ba6 c9             ret
               ;
               sectran:
                    ;translate the sector given by bc using the
                    ;translate table given by de
4ba7 eb             xchg               ;hl=.trans
4ba8 Ø9             dad     b          ;hl=.trans(sector)
4ba9 6e             mov     l,m        ;l = trans(sector)
4baa 26ØØ           mvi     h,Ø        ;hl= trans(sector)
4bac c9             ret                ;with value in hl
               ;
               setdma: ;set dma address given by registers b and c
4bad 69             mov     l,c        ;low order address
4bae 6Ø             mov     h,b        ;high order address
4baf 22ed4c         shld    dmaad      ;save the address
4bb2                ds      1Øh        ;space for setting the dma addres
4bc2 c9             ret
               ;
               read:   ;perform read operation (usually this is similar
               ;       so we will allow space to set up read command, th
               ;       common code in write)
4bc3                ds      1Øh        ;set up read command
4bd3 c3e64b         jmp     waitio     ;to perform the actual i/o
               ;
               write:  ;perform a write operation
4bd6                ds      1Øh        ;set up write command
               ;
               waitio: ;enter here from read and write to perform the ac
               ;       operation.  return a ØØh in register a if the ope
               ;       properly, and Ø1h if an error occurs during the r
```

```
            ;
            ;       in this case, we have saved the disk number in 'd
            ;                       the track number in 'track' (Ø-76
            ;                       the sector number in 'sector' (1-
            ;                       the dma address in 'dmaad' (Ø-655
4be6                ds      256     ;space reserved for i/o drivers
4ce6 3eØ1           mvi     a,1     ;error condition
4ce8 c9             ret             ;replaced when filled-in
            ;
            ;       the remainder of the cbios is reserved uninitiali
            ;       data area, and does not need to be a part of the
            ;       system memory image (the space must be available,
            ;       however, between "begdat" and "enddat").
            ;
4ce9        track:  ds      2       ;two bytes for expansion
4ceb        sector: ds      2       ;two bytes for expansion
4ced        dmaad:  ds      2       ;direct memory address
4cef        diskno: ds      1       ;disk number Ø-15
            ;
            ;       scratch ram area for bdos use
4cfØ =      begdat  equ     $       ;beginning of data area
4cfØ        dirbf:  ds      128     ;scratch directory area
4d7Ø        all ØØ: ds      31      ;allocation vector Ø
4d8f        allØ1:  ds      31      ;allocation vector 1
4dae        allØ2:  ds      31      ;allocation vector 2
4dcd        allØ3:  ds      31      ;allocation vector 3
4dec        chkØØ:  ds      16      ;check vector Ø
4dfc        chkØ1:  ds      16      ;check vector 1
4eØc        chkØ2:  ds      16      ;check vector 2
4e1c        chkØ3:  ds      16      ;check vector 3
            ;
4e2c =      enddat  equ     $       ;end of data area
Ø13c =      datsiz  equ     $-begdat;size of data area
4e2c                end
```

APPENDIX D: A SKELETAL GETSYS/PUTSYS PROGRAM

```
                ;       combined getsys and putsys programs from Sec 4.
                ;       Start the programs at the base of the TPA

0100                    org     0100h

0014 =          msize   equ     20              ; size of cp/m in Kbytes

                ; "bias" is the amount to add to addresses for > 20k
                ;       (referred to as "b" throughout the text)

0000 =          bias    equ     (msize-20)*1024
3400 =          ccp     equ     3400h+bias
3c00 =          bdos    equ     ccp+0800h
4a00 =          bios    equ     ccp+1600h

                ;       getsys programs tracks 0 and 1 to memory at
                ;       3880h + bias

                ;       register                usage
                ;         a             (scratch register)
                ;         b             track count (0...76)
                ;         c             sector count (1...26)
                ;         d,e           (scratch register pair)
                ;         h,l           load address
                ;         sp            set to stack address

                gstart:                                 ; start of getsys
0100 318033             lxi     sp,ccp-0080h            ; convenient plac
0103 218033             lxi     h,ccp-0080h             ; set initial loa
0106 0600               mvi     b,0                     ; start with trac
                rd$trk:                                 ; read next track
0108 0e01               mvi     c,1                     ; each track star
                rd$sec:
010a cd0003             call    read$sec                ; get the next se
010d 118000             lxi     d,128                   ; offset by one s
0110 19                 dad     d                       ;   (hl=hl+128)
0111 0c                 inr     c                       ; next sector
0112 79                 mov     a,c                     ; fetch sector nu
0113 fe1b               cpi     27                      ;   and see if la
0115 da0a01             jc      rdsec                   ; <, do one more

                ; arrive here at end of track, move to next track

0118 04                 inr     b                       ; track = track+1
0119 78                 mov     a,b                     ; check for last
011a fe02               cpi     2                       ; track = 2 ?
011c da0801             jc      rd$trk                  ; <, do another

                ; arrive here at end of load, halt for lack of anything b

011f fb                 ei
0120 76                 hlt
```

```
              ;       putsys program, places memory image starting at
              ;       3880h + bias back to tracks 0 and 1
              ;       start this program at the next page boundary

0200                  org     ($+0100h) and 0ff00h

              put$sys:
0200 318033           lxi     sp,ccp-0080h              ; convenient plac
0203 218033           lxi     h,ccp-0080h               ; start of dump
0206 0600             mvi     b,0                       ; start with trac
              wr$trk:
0208 0e01             mvi     c,1                       ; start with sect
              wr$sec:
020a cd0004           call    write$sec                 ; write one secto
020d 118000           lxi     d,128                     ; length of each
0210 19               dad     d                         ; <hl>=<hl> + 128
0211 0c               inr     c                         ; <c> = <c> + 1
0212 79               mov     a,c                       ; see if
0213 fe1b             cpi     27                        ;   past end of t
0215 da0a02           jc      wr$sec                    ; no, do another

              ;  arrive here at end of track, move to next track

0218 04               inr     b                         ; track = track+1
0219 78               mov     a,b                       ; see if
021a fe02             cpi     2                         ;   last track
021c da0802           jc      wr$trk                    ; no, do another

              ;       done with putsys, halt for lack of anything bette

021f fb               ei
0220 76               hlt


              ; user supplied subroutines for sector read and write

              ;       move to next page boundary

0300                  org     ($+0100h) and 0ff00h

              read$sec:
                      ; read the next sector
                      ; track in <b>,
                      ; sector in <c>
                      ; dmaaddr in <hl>

0300 c5               push    b
0301 e5               push    h

              ; user defined read operation goes here
0302                  ds      64

0342 e1               pop     h
0343 c1               pop     b
```

```
0344 c9                ret

0400                   org     ($+0100h) and 0ff00h    ; another page bo

            write$sec:

                       ; same parameters as read$sec

0400 c5                push    b
0401 e5                push    h

            ; user defined write operation goes here
0402                   ds      64

0442 el                pop     h
0443 cl                pop     b
0444 c9                ret

            ; end of getsys/putsys program

0445                   end
```

APPENDIX E: A SKELETAL COLD START LOADER

```
                ; this is a sample cold start loader which, when modified
                ; resides on track ØØ, sector Ø1 (the first sector on the
                ; diskette).  we assume that the controller has loaded
                ; this sector into memory upon system start-up (this pro-
                ; gram can be keyed-in, or can exist in read/only memory
                ; beyond the address space of the cp/m version you are
                ; running).  the cold start loader brings the cp/m system
                ; into memory at "loadp" (34ØØh + "bias").  in a 2Øk
                ; memory system, the value of "bias" is ØØØØh, with large
                ; values for increased memory sizes (see section 2). afte
                ; loading the cp/m system, the clod start loader branches
                ; to the "boot" entry point of the bios, which begins at
                ; "bios" + "bias."  the cold start loader is not used un-
                ; til the system is powered up again, as long as the bios
                ; is not overwritten.  the origin is assumed at ØØØØh, an
                ; must be changed if the controller brings the cold start
                ; loader into another area, or if a read/only memory area
                ; is used.

ØØØØ                    org     Ø               ; base of ram in cp/m

ØØ14 =          msize   equ     2Ø              ; min mem size in kbytes

ØØØØ =          bias    equ     (msize-2Ø)*1Ø24 ; offset from 2Øk system
34ØØ =          ccp     equ     34ØØh+bias      ; base of the ccp
4aØØ =          bios    equ     ccp+16ØØh       ; base of the bios
Ø3ØØ =          biosl   equ     Ø3ØØh           ; length of the bios
4aØØ =          boot    equ     bios
19ØØ =          size    equ     bios+biosl-ccp  ; size of cp/m system
ØØ32 =          sects   equ     size/128        ; # of sectors to load

                ;       begin the load operation

                cold:
ØØØØ Ø1Ø2ØØ             lxi     b,2             ; b=Ø, c=sector 2
ØØØ3 1632               mvi     d,sects         ; d=# sectors to load
ØØØ5 21ØØ34             lxi     h,ccp           ; base transfer address

                lsect:  ; load the next sector

                ;       insert inline code at this point to
                ;       read one 128 byte sector from the
                ;       track given in register b, sector
                ;       given in register c,
                ;       into the address given by <hl>
                ;
                ; branch to location "cold" if a read error occurs
```

```
             ;        ************************************************
             ;        *
             ;        *       user supplied read operation goes here...
             ;        *
             ;        ************************************************

ØØØ8 c36bØØ           jmp     past$patch        ; remove this when patche
ØØØb                  ds      6Øh

             past$patch:
             ; go to next sector if load is incomplete
ØØ6b 15               dcr     d                 ; sects=sects-1
ØØ6c caØØ4a           jz      boot              ; head for the bios

             ;        more sectors to load
             ;
             ; we aren't using a stack, so use <sp> as scratch registe
             ;        to hold the load address increment

ØØ6f 318ØØØ           lxi     sp,128            ; 128 bytes per sector
ØØ72 39               dad     sp                ; <hl> = <hl> + 128

ØØ73 Øc               inr     c                 ; sector = sector + 1
ØØ74 79               mov     a,c
ØØ75 fe1b             cpi     27                ; last sector of track?
ØØ77 daØ8ØØ           jc      lsect             ; no, go read another

             ; end of track, increment to next track

ØØ7a ØeØ1             mvi     c,1               ; sector = 1
ØØ7c Ø4               inr     b                 ; track = track + 1
ØØ7d c3Ø8ØØ           jmp     lsect             ; for another group
ØØ8Ø                  end                       ; of boot loader
```

```
;       CP/M 2.Ø disk re-definition library
;
;       Copyright (c) 1979
;       Digital Research
;       Box 579
;       Pacific Grove, CA
;       9395Ø
;
;       CP/M logical disk drives are defined using the
;       macros given below, where the sequence of calls
;       is:
;
;       disks   n
;       diskdef parameter-list-Ø
;       diskdef parameter-list-1
;       ...
;       diskdef parameter-list-n
;       endef
;
;       where n is the number of logical disk drives attached
;       to the CP/M system, and parameter-list-i defines the
;       characteristics of the ith drive (i=Ø,1,...,n-1)
;
;       each parameter-list-i takes the form
;               dn,fsc,lsc,[skf],bls,dks,dir,cks,ofs,[Ø]
;       where
;       dn      is the disk number Ø,1,...,n-1
;       fsc     is the first sector number (usually Ø or 1)
;       lsc     is the last sector number on a track
;       skf     is optional "skew factor" for sector translate
;       bls     is the data block size (1Ø24,2Ø48,...,16384)
;       dks     is the disk size in bls increments (word)
;       dir     is the number of directory elements (word)
;       cks     is the number of dir elements to checksum
;       ofs     is the number of tracks to skip (word)
;       [Ø]     is an optional Ø which forces 16K/directory en
;
;       for convenience, the form
;               dn,dm
;       defines disk dn as having the same characteristics as
;       a previously defined disk dm.
;
;       a standard four drive CP/M system is defined by
;               disks   4
;               diskdef Ø,1,26,6,1Ø24,243,64,64,2
;       dsk     set     Ø
;               rept    3
;       dsk     set     dsk+1
;               diskdef %dsk,Ø
;               endm
;               endef
;
;       the value of "begdat" at the end of assembly defines t
```

```
;       beginning of the uninitialize ram area above the bios,
;       while the value of "enddat" defines the next location
;       following the end of the data area.  the size of this
;       area is given by the value of "datsiz" at the end of t
;       assembly.  note that the allocation vector will be qui
;       large if a large disk size is defined with a small blo
;       size.
;
dskhdr  macro   dn
;;      define a single disk header list
dpe&dn: dw      xlt&dn,0000h    ;translate table
        dw      0000h,0000h     ;scratch area
        dw      dirbuf,dpb&dn   ;dir buff,parm block
        dw      csv&dn,alv&dn   ;check, alloc vectors
        endm
;
disks   macro   nd
;;      define nd disks
ndisks  set     nd      ;;for later reference
dpbase  equ     $       ;base of disk parameter blocks
;;      generate the nd elements
dsknxt  set     0
        rept    nd
        dskhdr  %dsknxt
dsknxt  set     dsknxt+1
        endm
        endm
;
dpbhdr  macro   dn
dpb&dn  equ     $               ;disk parm block
        endm
;
ddb     macro   data,comment
;;      define a db statement
        db      data            comment
        endm
;
ddw     macro   data,comment
;;      define a dw statement
        dw      data            comment
        endm
;
gcd     macro   m,n
;;      greatest common divisor of m,n
;;      produces value gcdn as result
;;      (used in sector translate table generation)
gcdm    set     m       ;;variable for m
gcdn    set     n       ;;variable for n
gcdr    set     0       ;;variable for r
        rept    65535
gcdx    set     gcdm/gcdn
gcdr    set     gcdm - gcdx*gcdn
        if      gcdr = 0
        exitm
        endif
```

```
gcdm    set     gcdn
gcdn    set     gcdr
        endm
        endm
;
diskdef macro   dn,fsc,lsc,skf,bls,dks,dir,cks,ofs,k16
;;      generate the set statements for later tables
        if      nul lsc
;;      current disk dn same as previous fsc
dpb&dn  equ     dpb&fsc ;equivalent parameters
als&dn  equ     als&fsc ;same allocation vector size
css&dn  equ     css&fsc ;same checksum vector size
xlt&dn  equ     xlt&fsc ;same translate table
        else
secmax  set     lsc-(fsc)       ;;sectors 0...secmax
sectors set     secmax+1;;number of sectors
als&dn  set     (dks)/8 ;;size of allocation vector
        if      ((dks) mod 8) ne 0
als&dn  set     als&dn+1
        endif
css&dn  set     (cks)/4 ;;number of checksum elements
;;      generate the block shift value
blkval  set     bls/128 ;;number of sectors/block
blkshf  set     0       ;;counts right 0's in blkval
blkmsk  set     0       ;;fills with 1's from right
        rept    16      ;;once for each bit position
        if      blkval=1
        exitm
        endif
;;      otherwise, high order 1 not found yet
blkshf  set     blkshf+1
blkmsk  set     (blkmsk shl 1) or 1
blkval  set     blkval/2
        endm
;;      generate the extent mask byte
blkval  set     bls/1024        ;;number of kilobytes/block
extmsk  set     0       ;;fill from right with 1's
        rept    16
        if      blkval=1
        exitm
        endif
;;      otherwise more to shift
extmsk  set     (extmsk shl 1) or 1
blkval  set     blkval/2
        endm
;;      may be double byte allocation
        if      (dks) > 256
extmsk  set     (extmsk shr 1)
        endif
;;      may be optional [0] in last position
        if      not nul k16
extmsk  set     k16
        endif
;;      now generate directory reservation bit vector
dirrem  set     dir     ;;# remaining to process
```

```
dirbks  set     bls/32  ;;number of entries per block
dirblk  set     0       ;;fill with 1's on each loop
        rept    16
        if      dirrem=0
        exitm
        endif
;;      not complete, iterate once again
;;      shift right and add 1 high order bit
dirblk  set     (dirblk shr 1) or 8000h
        if      dirrem > dirbks
dirrem  set     dirrem-dirbks
        else
dirrem  set     0
        endif
        endm
        dpbhdr  dn      ;;generate equ $
        ddw     %sectors,<;sec per track>
        ddb     %blkshf,<;block shift>
        ddb     %blkmsk,<;block mask>
        ddb     %extmsk,<;extnt mask>
        ddw     %(dks)-1,<;disk size-1>
        ddw     %(dir)-1,<;directory max>
        ddb     %dirblk shr 8,<;alloc0>
        ddb     %dirblk and 0ffh,<;alloc1>
        ddw     %(cks)/4,<;check size>
        ddw     %ofs,<;offset>
;;      generate the translate table, if requested
        if      nul skf
xlt&dn  equ     0               ;no xlate table
        else
        if      skf = 0
xlt&dn  equ     0               ;no xlate table
        else
;;      generate the translate table
nxtsec  set     0       ;;next sector to fill
nxtbas  set     0       ;;moves by one on overflow
        gcd     %sectors,skf
;;      gcdn = gcd(sectors,skew)
neltst  set     sectors/gcdn
;;      neltst is number of elements to generate
;;      before we overlap previous elements
nelts   set     neltst  ;;counter
xlt&dn  equ     $               ;translate table
        rept    sectors ;;once for each sector
        if      sectors < 256
        ddb     %nxtsec+(fsc)
        else
        ddw     %nxtsec+(fsc)
        endif
nxtsec  set     nxtsec+(skf)
        if      nxtsec >= sectors
nxtsec  set     nxtsec-sectors
        endif
nelts   set     nelts-1
        if      nelts = 0
```

```
nxtbas  set     nxtbas+1
nxtsec  set     nxtbas
nelts   set     neltst
        endif
        endm
        endif   ;;end of nul fac test
        endif   ;;end of nul bls test
        endm
;
defds   macro   lab,space
lab:    ds      space
        endm
;
lds     macro   lb,dn,val
        defds   lb&dn,%val&dn
        endm
;
endef   macro
;;      generate the necessary ram data areas
begdat  equ     $
dirbuf: ds      128     ;directory access buffer
dsknxt  set     0
        rept    ndisks  ;;once for each disk
        lds     alv,%dsknxt,als
        lds     csv,%dsknxt,css
dsknxt  set     dsknxt+1
        endm
enddat  equ     $
datsiz  equ     $-begdat
;;      db 0 at this point forces hex record
        endm
```

APPENDIX G: BLOCKING AND DEBLOCKING ALGORITHMS.

```
;******************************************************
;*                                                    *
;*      Sector Deblocking Algorithms for CP/M 2.0     *
;*                                                    *
;******************************************************
;
;       utility macro to compute sector mask
smask   macro   hblk
;;      compute log2(hblk), return @x as result
;;      (2 ** @x = hblk on return)
@y      set     hblk
@x      set     0
;;      count right shifts of @y until = 1
        rept    8
        if      @y = 1
        exitm
        endif
;;      @y is not 1, shift right one position
@y      set     @y shr 1
@x      set     @x + 1
        endm
        endm
;
;******************************************************
;*                                                    *
;*         CP/M to host disk constants                *
;*                                                    *
;******************************************************
blksiz  equ     2048            ;CP/M allocation size
hstsiz  equ     512             ;host disk sector size
hstspt  equ     20              ;host disk sectors/trk
hstblk  equ     hstsiz/128      ;CP/M sects/host buff
cpmspt  equ     hstblk * hstspt ;CP/M sectors/track
secmsk  equ     hstblk-1        ;sector mask
        smask   hstblk          ;compute sector mask
secshf  equ     @x              ;log2(hstblk)
;
;******************************************************
;*                                                    *
;*        BDOS constants on entry to write            *
;*                                                    *
;******************************************************
wrall   equ     0               ;write to allocated
wrdir   equ     1               ;write to directory
wrual   equ     2               ;write to unallocated
;
;******************************************************
;*                                                    *
;*      The BDOS entry points given below show the    *
;*      code which is relevant to deblocking only.    *
;*                                                    *
;******************************************************
;
```

```
;       DISKDEF macro, or hand coded tables go here
dpbase  equ     $               ;disk param block base
;
boot:
wboot:
        ;enter here on system boot to initialize
        xra     a               ;0 to accumulator
        sta     hstact          ;host buffer inactive
        sta     unacnt          ;clear unalloc count
        ret
;
seldsk:
        ;select disk
        mov     a,c             ;selected disk number
        sta     sekdsk          ;seek disk number
        mov     l,a             ;disk number to HL
        mvi     h,0
        rept    4               ;multiply by 16
        dad     h
        endm
        lxi     d,dpbase        ;base of parm block
        dad     d               ;hl=.dpb(curdsk)
        ret
;
settrk:
        ;set track given by registers BC
        mov     h,b
        mov     l,c
        shld    sektrk          ;track to seek
        ret
;
setsec:
        ;set sector given by register c
        mov     a,c
        sta     seksec          ;sector to seek
        ret
;
setdma:
        ;set dma address given by BC
        mov     h,b
        mov     l,c
        shld    dmaadr
        ret
;
sectran:
        ;translate sector number BC
        mov     h,b
        mov     l,c
        ret
;
```

```
;*******************************************************
;*                                                     *
;*      The READ entry point takes the place of        *
;*      the previous BIOS defintion for READ.          *
;*                                                     *
;*******************************************************
read:
        ;read the selected CP/M sector
        mvi     a,1
        sta     readop          ;read operation
        sta     rsflag          ;must read data
        mvi     a,wrual
        sta     wrtype          ;treat as unalloc
        jmp     rwoper          ;to perform the read
;
;*******************************************************
;*                                                     *
;*      The WRITE entry point takes the place of       *
;*      the previous BIOS defintion for WRITE.         *
;*                                                     *
;*******************************************************
write:
        ;write the selected CP/M sector
        xra     a               ;0 to accumulator
        sta     readop          ;not a read operation
        mov     a,c             ;write type in c
        sta     wrtype
        cpi     wrual           ;write unallocated?
        jnz     chkuna          ;check for unalloc
;
;       write to unallocated, set parameters
        mvi     a,blksiz/128    ;next unalloc recs
        sta     unacnt
        lda     sekdsk          ;disk to seek
        sta     unadsk          ;unadsk = sekdsk
        lhld    sektrk
        shld    unatrk          ;unatrk = sectrk
        lda     seksec
        sta     unasec          ;unasec = seksec
;
chkuna:
        ;check for write to unallocated sector
        lda     unacnt          ;any unalloc remain?
        ora     a
        jz      alloc           ;skip if not
;
;       more unallocated records remain
        dcr     a               ;unacnt = unacnt-1
        sta     unacnt
        lda     sekdsk          ;same disk?
        lxi     h,unadsk
        cmp     m               ;sekdsk = unadsk?
        jnz     alloc           ;skip if not
;
;       disks are the same
```

```
        lxi     h,unatrk
        call    sektrkcmp       ;sektrk = unatrk?
        jnz     alloc           ;skip if not
;
;       tracks are the same
        lda     seksec          ;same sector?
        lxi     h,unasec
        cmp     m               ;seksec = unasec?
        jnz     alloc           ;skip if not
;
;       match, move to next sector for future ref
        inr     m               ;unasec = unasec+1
        mov     a,m             ;end of track?
        cpi     cpmspt          ;count CP/M sectors
        jc      noovf           ;skip if no overflow
;
;       overflow to next track
        mvi     m,0             ;unasec = 0
        lhld    unatrk
        inx     h
        shld    unatrk          ;unatrk = unatrk+1
;
noovf:
        ;match found, mark as unnecessary read
        xra     a               ;0 to accumulator
        sta     rsflag          ;rsflag = 0
        jmp     rwoper          ;to perform the write
;
alloc:
        ;not an unallocated record, requires pre-read
        xra     a               ;0 to accum
        sta     unacnt          ;unacnt = 0
        inr     a               ;1 to accum
        sta     rsflag          ;rsflag = 1
;
;*****************************************************
;*                                                   *
;*      Common code for READ and WRITE follows       *
;*                                                   *
;*****************************************************
rwoper:
        ;enter here to perform the read/write
        xra     a               ;zero to accum
        sta     erflag          ;no errors (yet)
        lda     seksec          ;compute host sector
        rept    secshf
        ora     a               ;carry = 0
        rar                     ;shift right
        endm
        sta     sekhst          ;host sector to seek
;
;       active host sector?
        lxi     h,hstact        ;host active flag
        mov     a,m
        mvi     m,1             ;always becomes 1
```

```
        ora     a               ;was it already?
        jz      filhst          ;fill host if not
;
;       host buffer active, same as seek buffer?
        lda     sekdsk
        lxi     h,hstdsk        ;same disk?
        cmp     m               ;sekdsk = hstdsk?
        jnz     nomatch
;
;       same disk, same track?
        lxi     h,hsttrk
        call    sektrkcmp       ;sektrk = hsttrk?
        jnz     nomatch
;
;       same disk, same track, same buffer?
        lda     sekhst
        lxi     h,hstsec        ;sekhst = hstsec?
        cmp     m
        jz      match           ;skip if match
;
nomatch:
        ;proper disk, but not correct sector
        lda     hstwrt          ;host written?
        ora     a
        cnz     writehst        ;clear host buff
;
filhst:
        ;may have to fill the host buffer
        lda     sekdsk
        sta     hstdsk
        lhld    sektrk
        shld    hsttrk
        lda     sekhst
        sta     hstsec
        lda     rsflag          ;need to read?
        ora     a
        cnz     readhst         ;yes, if 1
        xra     a               ;0 to accum
        sta     hstwrt          ;no pending write
;
match:
        ;copy data to or from buffer
        lda     seksec          ;mask buffer number
        ani     secmsk          ;least signif bits
        mov     l,a             ;ready to shift
        mvi     h,0             ;double count
        rept    7               ;shift left 7
        dad     h
        endm
;       hl has relative host buffer address
        lxi     d,hstbuf
        dad     d               ;hl = host address
        xchg                    ;now in DE
        lhld    dmaadr          ;get/put CP/M data
        mvi     c,128           ;length of move
```

```
        lda     readop          ;which way?
        ora     a
        jnz     rwmove          ;skip if read
;
;       write operation, mark and switch direction
        mvi     a,1
        sta     hstwrt          ;hstwrt = 1
        xchg                    ;source/dest swap
;
rwmove:
        ;C initially 128, DE is source, HL is dest
        ldax    d               ;source character
        inx     d
        mov     m,a             ;to dest
        inx     h
        dcr     c               ;loop 128 times
        jnz     rwmove
;
;       data has been moved to/from host buffer
        lda     wrtype          ;write type
        cpi     wrdir           ;to directory?
        lda     erflag          ;in case of errors
        rnz                     ;no further processing
;
;       clear host buffer for directory write
        ora     a               ;errors?
        rnz                     ;skip if so
        xra     a               ;0 to accum
        sta     hstwrt          ;buffer written
        call    writehst
        lda     erflag
        ret
;
;*****************************************************
;*                                                   *
;*      Utility subroutine for 16-bit compare        *
;*                                                   *
;*****************************************************
sektrkcmp:
        ;HL = .unatrk or .hsttrk, compare with sektrk
        xchg
        lxi     h,sektrk
        ldax    d               ;low byte compare
        cmp     m               ;same?
        rnz                     ;return if not
;       low bytes equal, test high 1s
        inx     d
        inx     h
        ldax    d
        cmp     m       ;sets flags
        ret
;
```

```
;*****************************************************
;*                                                   *
;*      WRITEHST performs the physical write to      *
;*      the host disk, READHST reads the physical    *
;*      disk.                                        *
;*                                                   *
;*****************************************************
writehst:
        ;hstdsk = host disk #, hsttrk = host track #,
        ;hstsec = host sect #. write "hstsiz" bytes
        ;from hstbuf and return error flag in erflag.
        ;return erflag non-zero if error
        ret
;
readhst:
        ;hstdsk = host disk #, hsttrk = host track #,
        ;hstsec = host sect #. read "hstsiz" bytes
        ;into hstbuf and return error flag in erflag.
        ret
;
;*****************************************************
;*                                                   *
;*      Unitialized RAM data areas                   *
;*                                                   *
;*****************************************************
;
sekdsk: ds      1               ;seek disk number
sektrk: ds      2               ;seek track number
seksec: ds      1               ;seek sector number
;
hstdsk: ds      1               ;host disk number
hsttrk: ds      2               ;host track number
hstsec: ds      1               ;host sector number
;
sekhst: ds      1               ;seek shr secshf
hstact: ds      1               ;host active flag
hstwrt: ds      1               ;host written flag
;
unacnt: ds      1               ;unalloc rec cnt
unadsk: ds      1               ;last unalloc disk
unatrk: ds      2               ;last unalloc track
unasec: ds      1               ;last unalloc sector
;
erflag: ds      1               ;error reporting
rsflag: ds      1               ;read sector flag
readop: ds      1               ;1 if read operation
wrtype: ds      1               ;write operation type
dmaadr: ds      2               ;last dma address
hstbuf: ds      hstsiz          ;host buffer
;
```

```
;*****************************************************
;*                                                   *
;*      The ENDEF macro invocation goes here         *
;*                                                   *
;*****************************************************
        end
```