SIEMENS

Betriebssystem CP/M-86®

Systembeschreibung (System Guide)

COPYRIGHT

DISCLAIMER

TRADEMARKS

The "CP/M-86 System Guide" was prepared using the Digital Research TEX-80™ Text Formatter and printed in the United States of America by Commercial Press/Monterey.

```
**********************************
*  Second Printing:  June 1981   *
**********************************
```

Foreword

The CP/M-86 System Guide presents the system programming aspects of CP/M-86®, a single-user operating system for the Intel 8086 and 8088 16-bit microprocessors. The discussion assumes the reader is familiar with CP/M the Digital Research 8-bit operating system. To clarify specific differences with CP/M-86, this document refers to the 8-bit version of CP/M as CP/M-80™. Elements common to both systems are simply called CP/M features.

CP/M-80 and CP/M-86 are equivalent at the user interface level and thus the Digital Research documents:

- An Introduction to CP/M Features and Facilities
- ED: A Context Editor for the CP/M Disk System
- CP/M 2 User's Guide

are shipped with the CP/M-86 package. Also included is the CP/M-86 Programmer's Guide, which describes ASM-86™ and DDT-86™, Digital Research's 8086 assembler and interactive debugger.

This System Guide presents an overview of the CP/M-86 programming interface conventions. It also describes procedures for adapting CP/M-86 to a custom hardware enviornment. This information parallels that presented in the CP/M 2 Interface Guide and the CP/M 2 Alteration Guide.

Section 1 gives an overview of CP/M-86 and summarizes its differences with CP/M-80. Section 2 describes the general execution environment while Section 3 tells how to generate command files. Sections 4 and 5 respectively define the programming interfaces to the Basic Disk Operating System and the Basic Input/Output System. Section 6 discusses alteration of the BIOS to support custom disk configurations, and Section 7 describes the loading operation and the organization of the CP/M-86 system file.

Table of Contents

Appendixes

Section 1
CP/M-86 System Overview

1.1 CP/M-86 General Characteristics

CP/M-86 contains all facilities of CP/M-80 with additional features to account for increased processor address space of up to a megabyte (1,048,576) of main memory. Further, CP/M-86 maintains file compatibility with all previous versions of CP/M. The file structure of version 2 of CP/M is used, allowing as many as sixteen drives with up to eight megabytes on each drive. Thus, CP/M-80 and CP/M-86 systems may exchange files without modifying the file format.

CP/M-86 resides in the file CPM.SYS, which is loaded into memory by a cold start loader during system initialization. The cold start loader resides on the first two tracks of the system disk. CPM.SYS contains three program modules: the Console Command Processor (CCP), the Basic Disk Operating System (BDOS), and the user-configurable Basic I/O System (BIOS). The CCP and BDOS portions occupy approximately 10K bytes, while the size of the BIOS varies with the implementation. The operating system executes in any portion of memory above the reserved interrupt locations, while the remainder of the address space is partitioned into as many as eight non-contiguous regions, as defined in a BIOS table. Unlike CP/M-80, the CCP area cannot be used as a data area subsequent to transient program load; all CP/M-86 modules remain in memory at all times, and are not reloaded at a warm start.

Similar to CP/M-80, CP/M-86 loads and executes memory image files from disk. Memory image files are preceded by a "header record," defined in this document, which provides information required for proper program loading and execution. Memory image files under CP/M-86 are identified by a "CMD" file type.

Unlike CP/M-80, CP/M-86 does not use absolute locations for system entry or default variables. The BDOS entry takes place through a reserved software interrupt, while entry to the BIOS is provided by a new BDOS call. Two variables maintained in low memory under CP/M-80, the default disk number and I/O Byte, are placed in the CCP and BIOS, respectively. Dependence upon absolute addresses is minimized in CP/M-86 by maintaining initial "base page" values, such as the default FCB and default command buffer, in the transient program data area.

Utility programs such as ED, PIP, STAT and SUBMIT operate in the same manner under CP/M-86 and CP/M-80. In its operation, DDT-86 resembles DDT supplied with CP/M-80. It allows interactive debugging of 8086 and 8088 machine code. Similarly, ASM-86 allows assembly language programming and development for the 8086 and 8088 using Intel-like mnemonics.

The GENCMD (Generate CMD) utility replaces the LOAD program of CP/M-80, and converts the hex files produced by ASM-86 or Intel utilities into memory image format suitable for execution under CP/M-86. Further, the LDCOPY (Loader Copy) program replaces SYSGEN, and is used to copy the cold start loader from a system disk for replication. In addition, a variation of GENCMD, called LMCMD, converts output from the Intel LOC86 utility into CMD format. Finally, GENDEF (Generate DISKDEF) is provided as an aid in producing custom disk parameter tables. ASM-86, GENCMD, LMCMD, and GENDEF are also supplied in "COM" file format for cross-development under CP/M-80.

Several terms used throughout this manual are defined in Table 1-1 below:

Table 1-1. CP/M-86 Terms

Term	Meaning
Nibble	4-bit half-byte
Byte	8-bit value
Word	16-bit value
Double Word	32-bit value
Paragraph	16 contiguous bytes
Paragraph Boundary	An address divisible evenly by 16 (low order nibble 0)
Segment	Up to 64K contiguous bytes
Segment Register	One of CS, DS, ES, or SS
Offset	16-bit displacement from a segment register
Group	A segment-register-relative relocatable program unit
Address	The effective memory address derived from the composition of a segment register value with an offset value

A group consists of segments that are loaded into memory as a single unit. Since a group may consist of more than 64K bytes, it is the responsibility of the application program to manage segment registers when code or data beyond the first 64K segment is accessed.

CP/M-86 supports eight program groups: the code, data, stack and extra groups as well as four auxiliary groups. When a code, data, stack or extra group is loaded, CP/M-86 sets the respective segment register (CS, DS, SS or ES) to the base of the group. CP/M-86 can also load four auxiliary groups. A transient program manages the location of the auxiliary groups using values stored by CP/M-86 in the user's base page.

1.2 CP/M-80 and CP/M-86 Differences

The structure of CP/M-86 is as close to CP/M-80 as possible in order to provide a familiar programming environment which allows application programs to be transported to the 8086 and 8088 processors with minimum effort. This section points out the specific differences between CP/M-80 and CP/M-86 in order to reduce your time in scanning this manual if you are already familiar with CP/M-80. The terms and concepts presented in this section are explained in detail throughout this manual, so you will need to refer to the Table of Contents to find relevant sections which provide specific definitions and information.

Due to the nature of the 8086 processor, the fundamental difference between CP/M-80 and CP/M-86 is found in the management of the various relocatable groups. Although CP/M-80 references absolute memory locations by necessity, CP/M-86 takes advantage of the static relocation inherent in the 8086 processor. The operating system itself is usually loaded directly above the interrupt locations, at location 0400H, and relocatable transient programs load in the best fit memory region. However, you can load CP/M-86 into any portion of memory without changing the operating system (thus, there is no MOVCPM utility with CP/M-86), and transient programs will load and run in any non-reserved region.

Three general memory models are presented below, but if you are converting 8080 programs to CP/M-86, you can use either the 8080 Model or Small Model and leave the Compact Model for later when your addressing needs increase. You'll use GENCMD, described in Section 3.2, to produce an executable program file from a hex file. GENCMD parameters allow you to specify which memory model your program requires.

CP/M-86 itself is constructed as an 8080 Model. This means that all the segment registers are placed at the base of CP/M-86, and your customized BIOS is identical, in most respects, to that of CP/M-80 (with changes in instruction mnemonics, of course). In fact, the only additions are found in the SETDMAB, GETSEGB, SETIOB, and GETIOB entry points in the BIOS. Your warm start subroutine is simpler since you are not required to reload the CCP and BDOS under CP/M-86. One other point: if you implement the IOBYTE facility, you'll have to define the variable in your BIOS. Taking these changes into account, you need only perform a simple translation of your CP/M-80 BIOS into 8086 code in order to implement your 8086 BIOS.

If you've implemented CP/M-80 Version 2, you already have disk definition tables which will operate properly with CP/M-86. You may wish to attach different disk drives, or experiment with sector skew factors to increase performance. If so, you can use the new GENDEF utility which performs the same function as the DISKDEF macro used by MAC under CP/M-80. You'll find, however, that GENDEF provides you with more information and checks error conditions better than the DISKDEF macro.

Although generating a CP/M-86 system is generally easier than generating a CP/M-80 system, complications arise if you are using single-density floppy disks. CP/M-86 is too large to fit in the two-track system area of a single-density disk, so the bootstrap operation must perform two steps to load CP/M-86: first the bootstrap must load the cold start loader, then the cold start loader loads CP/M-86 from a system file. The cold start loader includes a LDBIOS which is identical to your CP/M-86 BIOS with the exception of the INIT entry point. You can simplify the LDBIOS if you wish because the loader need not write to the disk. If you have a double-density disk or reserve enough tracks on a single-density disk, you can load CP/M-86 without a two-step boot.

To make a BDOS system call, use the reserved software interrupt #244. The jump to the BDOS at location 0005 found in CP/M-80 is not present in CP/M-86. However, the address field at offset 0006 is present so that programs which "size" available memory using this word value will operate without change. CP/M-80 BDOS functions use certain 8080 registers for entry parameters and returned values. CP/M-86 BDOS functions use a table of corresponding 8086 registers. For example, the 8086 registers CH and CL correspond to the 8080 registers B and C. Look through the list of BDOS function numbers in Table 4-2, and you'll find that functions 0, 27, and 31 have changed slightly. Several new functions have been added, but they do not affect existing programs.

One major philosophical difference is that in CP/M-80, all addresses sent to the BDOS are simply 16-bit values in the range 0000H to 0FFFFH. In CP/M-86, however, the addresses are really just 16-bit offsets from the DS (Data Segment) register which is set to the base of your data area. If you translate an existing CP/M-80 program to the CP/M-86 environment, your data segment will be less than 64K bytes. In this case, the DS register need not be changed following initial load, and thus all CP/M-80 addresses become simple DS-relative offsets in CP/M-86.

Under CP/M-80, programs terminate in one of three ways: by returning directly to the CCP, by calling BDOS function 0, or by transferring control to absolute location 0000H. CP/M-86, however, supports only the first two methods of program termination. This has the side effect of not providing the automatic disk system reset following the jump to 0000H which, instead, is accomplished by entering a CONTROL-C at the CCP level.

You'll find many new facilities in CP/M-86 that will simplify your programming and expand your application programming capability. But, we've designed CP/M-86 to make it easy to get started: in short, if you are converting from CP/M-80 to CP/M-86, there will be no major changes beyond the translation to 8086 machine code. Further, programs you design for CP/M-86 are upward compatible with MP/M-86, our multitasking operating system, as well as CP/NET-86 which provides a distributed operating system in a network environment.

Section 2
Command Setup and Execution Under CP/M-86

This section discusses the operation of the Console Command Processor (CCP), the format of transient programs, CP/M-86 memory models, and memory image formats.

2.1 CCP Built-in and Transient Commands

The operation of the CP/M-86 CCP is similar to that of CP/M-80. Upon initial cold start, the CP/M sign-on message is printed, drive A is automatically logged in, and the standard prompt is issued at the console. CP/M-86 then waits for input command lines from the console, which may include one of the built-in commands

DIR ERA REN TYPE USER

(note that SAVE is not supported under CP/M-86 since the equivalent function is performed by DDT-86).

Alternatively, the command line may begin with the name of a transient program with the assumed file type "CMD" denoting a "command file." The CMD file type differentiates transient command files used under CP/M-86 from COM files which operate under CP/M-80.

The CCP allows multiple programs to reside in memory, providing facilities for background tasks. A transient program such as a debugger may load additional programs for execution under its own control. Thus, for example, a background printer spooler could first be loaded, followed by an execution of DDT-86. DDT-86 may, in turn, load a test program for a debugging session and transfer control to the test program between breakpoints. CP/M-86 keeps account of the order in which programs are loaded and, upon encountering a CONTROL-C, discontinues execution of the most recent program activated at the CCP level. A CONTROL-C at the DDT-86 command level aborts DDT-86 and its test program. A second CONTROL-C at the CCP level aborts the background printer spooler. A third CONTROL-C resets the disk system. Note that program abort due to CONTROL-C does not reset the disk system, as is the case in CP/M-80. A disk reset does not occur unless the CONTROL-C occurs at the CCP command input level with no programs residing in memory.

When CP/M-86 receives a request to load a transient program from the CCP or another transient program, it checks the program's memory requirements. If sufficient memory is available, CP/M-86 assigns the required amount of memory to the program and loads the program. Once loaded, the program can request additional memory from the BDOS for buffer space. When the program is terminated, CP/M-86 frees both the program memory area and any additional buffer space.

2.2 Transient Program Execution Models

The initial values of the segment registers are determined by one of three "memory models" used by the transient program, and described in the CMD file header. The three memory models are summarized in Table 2-1 below.

Table 2-1. CP/M-86 Memory Models

Model	Group Relationships
8080 Model	Code and Data Groups Overlap
Small Model	Independent Code and Data Groups
Compact Model	Three or More Independent Groups

The 8080 Model supports programs which are directly translated from CP/M-80 when code and data areas are intermixed. The 8080 model consists of one group which contains all the code, data, and stack areas. Segment registers are initialized to the starting address of the region containing this group. The segment registers can, however, be managed by the application program during execution so that multiple segments within the code group can be addressed.

The Small Model is similar to that defined by Intel, where the program consists of an independent code group and a data group. The Small Model is suitable for use by programs taken from CP/M-80 where code and data is easily separated. Note again that the code and data groups often consist of, but are not restricted to, single 64K byte segments.

The Compact Model occurs when any of the extra, stack, or auxiliary groups are present in program. Each group may consist of one or more segments, but if any group exceeds one segment in size, or if auxiliary groups are present, then the application program must manage its own segment registers during execution in order to address all code and data areas.

The three models differ primarily in the manner in which segment registers are initialized upon transient program loading. The operating system program load function determines the memory model used by a transient program by examining the program group usage, as described in the following sections.

2.3 The 8080 Memory Model

The 8080 Model is assumed when the transient program contains only a code group. In this case, the CS, DS, and ES registers are initialized to the beginning of the code group, while the SS and SP registers remain set to a 96-byte stack area in the CCP. The Instruction Pointer Register (IP) is set to 100H, similar to CP/M-80, thus allowing base page values at the beginning of the code group. Following program load, the 8080 Model appears as shown in Figure 2-1, where low addresses are shown at the top of the diagram:

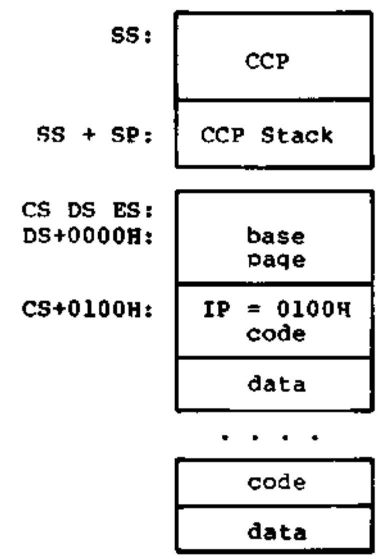

Figure 2-1. CP/M-86 8080 Memory Model

The intermixed code and data regions are indistinguishable. The "base page" values, described below, are identical to CP/M-80, allowing simple translation from 8080, 8085, or Z80 code into the 8086 and 8088 environment. The following ASM-86 example shows how to code an 8080 model transient program.

```
                eseg
                org     100h
                .
                .       (code)
        endcs   equ     $
                dseg
                org     offset endcs
                .
                .      (data)
                end
```

2.4 The Small Memory Model

The Small Model is assumed when the transient program contains both a code and data group. (In ASM-86, all code is generated following a CSEG directive, while data is defined following a DSEG directive with the origin of the data segment independent of the code segment.) In this model, CS is set to the beginning of the code group, the DS and ES are set to the start of the data group, and the SS and SP registers remain in the CCP's stack area as shown in Figure 2-2.

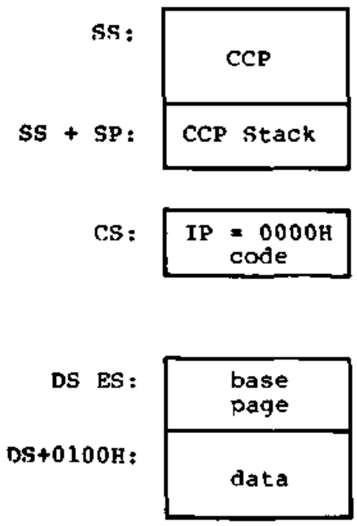

Figure 2-2. CP/M-86 Small Memory Model

The machine code begins at CS+0000H, the "base page" values begin at DS+0000H, and the data area starts at DS+0100H. The following ASM-86 example shows how to code a small model transient program.

```
cseg
.
.       (code)
dseg
org       100h
.
.       (data)
end
```

2.5 The Compact Memory Model

The Compact Model is assumed when code and data groups are present, along with one or more of the remaining stack, extra, or auxiliary groups. In this case, the CS, DS, and ES registers are set to the base addresses of their respective areas. Figure 2-3 shows the initial configuration of segment registers in the Compact Model. The values of the various segment registers can be programmatically changed during execution by loading from the initial values placed in base page by the CCP, thus allowing access to the entire memory space.

If the transient program intends to use the stack group as a stack area, the SS and SP registers must be set upon entry. The SS and SP registers remain in the CCP area, even if a stack group is defined. Although it may appear that the SS and SP registers should be set to address the stack group, there are two contradictions. First, the transient program may be using the stack group as a data area. In that case, the Far Call instruction used by the CCP to transfer control to the transient program could overwrite data in the stack area. Second, the SS register would logically be set to the base of the group, while the SP would be set to the offset of the end of the group. However, if the stack group exceeds 64K the address range from the base to the end of the group exceeds a 16-bit offset value.

The following ASM-86 example shows how to code a compact model transient program.

```
cseg
.
.       (code)
dseg
org     100h
.
.       (data)
eseg
.
.       (more data)
sseg
.
.       (stack area)
end
```

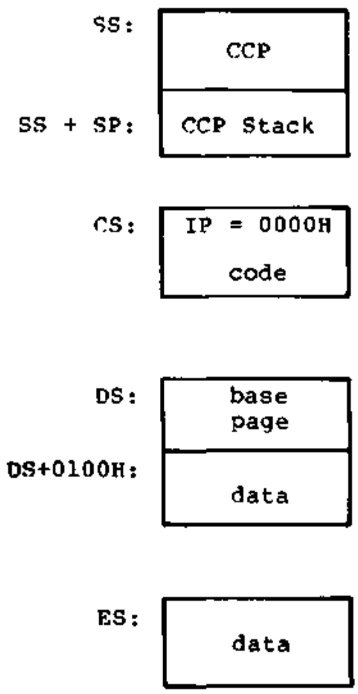

Figure 2-3. CP/M-86 Compact Memory Model

2.6 Base Page Initialization

Similar to CP/M-80, the CP/M-86 base page contains default values and locations initialized by the CCP and used by the transient program. The base page occupies the regions from offset 0000H through 00FFH relative to the DS register. The values in the base page for CP/M-86 include those of CP/M-80, and appear in the same relative positions, as shown in Figure 2-4.

DS + 0000:	LC0	LC1	LC2
DS + 0003:	BC0	BC1	M80
DS + 0006:	LD0	LD1	LD2
DS + 0009:	BD0	BD1	xxx
DS + 000C:	LE0	LE1	LE2
DS + 000F:	BE0	BE1	xxx
DS + 0012:	LS0	LS1	LS2
DS + 0015:	BS0	BS1	xxx
DS + 0018:	LX0	LX1	LX2
DS + 001B:	BX0	BX1	xxx
DS + 001E:	LX0	LX1	LX2
DS + 0021:	BX0	BX1	xxx
DS + 0024:	LX0	LX1	LX2
DS + 0027:	BX0	BX1	xxx
DS + 002A:	LX0	LX1	LX2
DS + 002D:	BX0	BX1	xxx
DS + 0030: . . . DS + 005B:	Not Currently Used		
DS + 005C:	Default FCB		
DS + 0080:	Default Buffer		
DS + 0100:	Begin User Data		

Figure 2-4. CP/M-86 Base Page Values

Each byte is indexed by 0, 1, and 2, corresponding to the standard Intel storage convention of low, middle, and high-order (most significant) byte. "xxx" in Figure 2-4 marks unused bytes. LC is the last code group location (24-bits, where the 4 high-order bits equal zero).

In the 8080 Model, the low order bytes of LC (LC0 and LC1) never exceed 0FFFFH and the high order byte (LC2) is always zero. BC is base paragraph address of the code group (16-bits). LD and BD provide the last position and paragraph base of the data group. The last position is one byte less than the group length. It should be noted that bytes LD0 and LD1 appear in the same relative positions of the base page in both CP/M-80 and CP/M-86, thus easing the program translation task. The M80 byte is equal to 1 when the 8080 Memory Model is in use. LE and BE provide the length and paragraph base of the optional extra group, while LS and BS give the optional stack group length and base. The bytes marked LX and BX correspond to a set of four optional independent groups which may be required for programs which execute using the Compact Memory Model. The initial values for these descriptors are derived from the header record in the memory image file, described in the following section.

2.7 Transient Program Load and Exit

Similar to CP/M-80, the CCP parses up to two filenames following the command and places the properly formatted FCB's at locations 005CH and 006CH in the base page relative to the DS register. Under CP/M-80, the default DMA address is initialized to 0080H in the base page. Due to the segmented memory of the 8086 and 8088 processors, the DMA address is divided into two parts: the DMA segment address and the DMA offset. Therefore, under CP/M-86, the default DMA base is initialized to the value of DS, and the default DMA offset is initialized to 0080H. Thus, CP/M-80 and CP/M-86 operate in the same way: both assume the default DMA buffer occupies the second half of the base page.

The CCP transfers control to the transient program through an 8086 "Far Call." The transient program may choose to use the 96-byte CCP stack and optionally return directly to the CCP upon program termination by executing a "Far Return." Program termination also occurs when BDOS function zero is executed. Note that function zero can terminate a program without removing the program from memory or changing the memory allocation state (see Section 4.2). The operator may terminate program execution by typing a single CONTROL-C during line edited input which has the same effect as the program executing BDOS function zero. Unlike the operation of CP/M-80, no disk reset occurs and the CCP and BDOS modules are not reloaded from disk upon program termination.

Section 3
Command (CMD) File Generation

As mentioned previously, two utility programs are provided with CP/M-86, called GENCMD and LMCMD, which are used to produce CMD memory image files suitable for execution under CP/M-86. GENCMD accepts Intel 8086 "hex" format files as input, while LMCMD reads Intel L-module files output from the standard Intel LOC86 Object Code Locator utility. GENCMD is used to process output from the Digital Research ASM-86 assembler and Intel's OH86 utility, while LMCMD is used when Intel compatible developmental software is available for generation of programs targeted for CP/M-86 operation.

3.1 Intel 8086 Hex File Format

GENCMD input is in Intel "hex" format produced by both the Digital Research ASM-86 assembler and the standard Intel OH86 utility program (see Intel document #9800639-03 entitled "MCS-86 Software Development Utitities Operating Instructions for ISIS-II Users"). The CMD file produced by GENCMD contains a header record which defines the memory model and memory size requirements for loading and executing the CMD file.

An Intel "hex" file consists of the traditional sequence of ASCII records in the following format:

:	l	l	a	a	a	a	t	t	d	d	d	. . .	d	c	c

where the beginning of the record is marked by an ASCII colon, and each subsequent digit position contains an ASCII hexadecimal digit in the range 0-9 or A-F. The fields are defined in Table 3-1.

Table 3-1. Intel Hex Field Definitions

Field	Contents
ll	Record Length 00-FF (0-255 in decimal)
aaaa	Load Address
tt	Record Type: 00 data record, loaded starting at offset aaaa from current base paragraph 01 end of file, cc = FF 02 extended address, aaaa is paragraph base for subsequent data records 03 start address is aaaa (ignored, IP set according to memory model in use) The following are output from ASM-86 only: 81 same as 00, data belongs to code segment 82 same as 00, data belongs to data segment 83 same as 00, data belongs to stack segment 84 same as 00, data belongs to extra segment 85 paragraph address for absolute code segment 86 paragraph address for absolute data segment 87 paragraph address for absolute stack segment 88 paragraph address for absolute extra segment
d	Data Byte
cc	Check Sum (00 - Sum of Previous Digits)

All characters preceding the colon for each record are ignored. (Additional hex file format information is included in the ASM-86 User's Guide, and in Intel's document #9800821A entitled "MCS-86 Absolute Object File Formats.")

3.2 Operation of GENCMD

The GENCMD utility is invoked at the CCP level by typing

```
GENCMD filename parameter-list
```

where the filename corresponds to the hex input file with an assumed (and unspecified) file type of H86. GENCMD accepts optional parameters to specifically identify the 8080 Memory Model and to describe memory requirements of each segment group. The GENCMD parameters are listed following the filename, as shown in the command line above where the parameter-list consists of a sequence of keywords and values separated by commas or blanks. The keywords are:

```
8080  CODE  DATA  EXTRA  STACK  X1  X2  X3  X4
```

The 8080 keyword forces a single code group so that the BDOS load function sets up the 8080 Memory Model for execution, thus allowing intermixed code and data within a single segment. The form of this command is

```
GENCMD filename 8080
```

The remaining keywords follow the filename or the 8080 option and define specific memory requirements for each segment group, corresponding one-to-one with the segment groups defined in the previous section. In each case, the values corresponding to each group are enclosed in square brackets and separated by commas. Each value is a hexadecimal number representing a paragraph address or segment length in paragraph units denoted by hhhh, prefixed by a single letter which defines the meaning of each value:

```
Ahhhh  Load the group at absolute location hhhh
Bhhhh  The group starts at hhhh in the hex file
Mhhhh  The group requires a minimum of hhhh * 16 bytes
Xhhhh  The group can address a maximum of hhhh * 16 bytes
```

Generally, the CMD file header values are derived directly from the hex file and the parameters shown above need not be included. The following situations, however, require the use of GENCMD parameters.

- The 8080 keyword is included whenever ASM-86 is used in the conversion of 8080 programs to the 8086/8088 environment when code and data are intermixed within a single 64K segment, regardless of the use of CSEG and DSEG directives in the source program.
- An absolute address (A value) must be given for any group which must be located at an absolute location. Normally, this value is not specified since CP/M-86 cannot generally ensure that the required memory region is available, in which case the CMD file cannot be loaded.
- The B value is used when GENCMD processes a hex file produced by Intel's OH86, or similar utility program that contains more than one group. The output from OH86 consists of a sequence of data records with no information to identify code, data, extra, stack, or auxiliary groups. In this case, the B value marks the beginning address of the group named by the keyword, causing GENCMD to load data following this address to the named group (see the examples below). Thus, the B value is normally used to mark the boundary between code and data segments when no segment information is included in the hex file. Files produced by ASM-86 do not require the use of the B value since segment information is included in the hex file.

- The minimum memory value (M value) is included only when the hex records do not define the minimum memory requirements for the named group. Generally, the code group size is determined precisely by the data records loaded into the area. That is, the total space required for the group is defined by the range between the lowest and highest data byte addresses. The data group, however, may contain uninitialized storage at the end of the group and thus no data records are present in the hex file which define the highest referenced data item. The highest address in the data group can be defined within the source program by including a "DB 0" as the last data item. Alternatively, the M value can be included to allocate the additional space at the end of the group. Similarly, the stack, extra, and auxiliary group sizes must be defined using the M value unless the highest addresses within the groups are implicitly defined by data records in the hex file.

- The maximum memory size, given by the X value, is generally used when additional free memory may be needed for such purposes as I/O buffers or symbol tables. If the data area size is fixed, then the X parameter need not be included. In this case, the X value is assumed to be the same as the M value. The value XFFFF allocates the largest memory region available but, if used, the transient program must be aware that a three-byte length field is produced in the base page for this group where the high order byte may be non-zero. Programs converted directly from CP/M-80 or programs that use a 2-byte pointer to address buffers should restrict this value to XFFF or less, producing a maximum allocation length of 0FFF0H bytes.

The following GENCMD command line transforms the file X.H86 into the file X.CMD with the proper header record:

```
gencmd x code[a40] data[m30,xfff]
```

In this case, the code group is forced to paragraph address 40H, or equivalently, byte address 400H. The data group requires a minimum of 300H bytes, but can use up to 0FFF0H bytes, if available.

Assuming a file Y.H86 exists on drive B containing Intel hex records with no interspersed segment information, the command

```
gencmd b:y data[b30,m20] extra[b50] stack[m40] x1[m40]
```

produces the file Y.CMD on drive B by selecting records beginning at address 0000H for the code segment, with records starting at 300H allocated to the data segment. The extra segment is filled from records beginning at 500H, while the stack and auxiliary segment #1 are uninitialized areas requiring a minimum of 400H bytes each. In this example, the data area requires a minimum of 200H bytes. Note again, that the B value need not be included if the Digital Research ASM-86 assembler is used.

3.3 Operation of LMCMD

The LMCMD utility operates in exactly the same manner as GENCMD, with the exception that LMCMD accepts an Intel L-module file as input. The primary advantage of the L-module format is that the file contains internally coded information which defines values which would otherwise be required as parameters to GENCMD, such the beginning address of the group's data segment. Currently, however, the only language processors which use this format are the standard Intel development packages, although various independent vendors will, most likely, take advantage of this format in the future.

3.4 Command (CMD) File Format

The CMD file produced by GENCMD and LMCMD consists of the 128-byte header record followed immediately by the memory image. Under normal circumstances, the format of the header record is of no consequence to a programmer. For completeness, however, the various fields of this record are shown in Figure 3-1.

←— 128 Bytes —→				
GD#1	GD#2	GD#3	GD#4	GD#5-GD#8. . .
Code, Data, Extra, Stack, Auxiliary				

Figure 3-1. CMD File Header Format

In Figure 3-1, GD#2 through GD#8 represent "Group Descriptors." Each Group Descriptor corresponds to an independently loaded program unit and has the following fields:

8-bit	16-bit	16-bit	16-bit	16-bit
G-Form	G-Length	A-Base	G-Min	G-Max

where G-Form describes the group format, or has the value zero if no more descriptors follow. If G-Form is non-zero, then the 8-bit value is parsed as two fields:

G-Form:

4-bit	4-bit
x x x x	G-Type

The G-Type field determines the Group Descriptor type. The valid Group Descriptors have a G-Type in the range 1 through 9, as shown in Table 3-2 below.

Table 3-2. Group Descriptors

G-Type	Group Type
1	Code Group
2	Data Group
3	Extra Group
4	Stack Group
5	Auxiliary Group #1
6	Auxiliary Group #2
7	Auxiliary Group #3
8	Auxiliary Group #4
9	Shared Code Group
10 - 14	Unused, but Reserved
15	Escape Code for Additional Types

All remaining values in the group descriptor are given in increments of 16-byte paragraph units with an assumed low-order 0 nibble to complete the 20-bit address. G-Length gives the number of paragraphs in the group. Given a G-length of 0080H, for example, the size of the group is 00800H = 2048D bytes. A-Base defines the base paragraph address for a non-relocatable group while G-Min and G-Max define the minimum and maximum size of the memory area to allocate to the group. G-Type 9 marks a "pure" code group for use under MP/M-86 and future versions of CP/M-86. Presently a Shared Code Group is treated as a non-shared Program Code Group under CP/M-86.

The memory model described by a header record is implicitly determined by the Group Descriptors. The 8080 Memory Model is assumed when only a code group is present, since no independent data group is named. The Small Model is implied when both a code and data group are present, but no additional group descriptors occur. Otherwise, the Compact Model is assumed when the CMD file is loaded.

Section 4
Basic Disk Operating System Functions

This section presents the interface conventions which allow transient program access to CP/M-86 BDOS and BIOS functions. The BDOS calls correspond closely to CP/M-80 Version 2 in order to simplify translation of existing CP/M-80 programs for operation under CP/M-86. BDOS entry and exit conditions are described first, followed by a presentation of the individual BDOS function calls.

4.1 BDOS Parameters and Function Codes

Entry to the BDOS is accomplished through the 8086 software interrupt #224, which is reserved by Intel Corporation for use by CP/M-86 and MP/M-86. The function code is passed in register CL with byte parameters in DL and word parameters in DX. Single byte values are returned in AL, word values in both AX and BX, and double word values in ES and BX. All segment registers, except ES, are saved upon entry and restored upon exit from the BDOS (corresponding to PL/M-86 conventions). Table 4-1 summarizes input and output parameter passing:

Table 4-1. BDOS Parameter Summary

BDOS Entry Registers	BDOS Return Registers
CL Function Code	Byte value returned in AL
DL Byte Parameter	Word value returned in both AX and BX
DX Word Parameter	Double-word value returned with offset in BX and segment in ES
DS Data Segment	

Note that the CP/M-80 BDOS requires an "information address" as input to various functions. This address usually provides buffer or File Control Block information used in the system call. In CP/M-86, however, the information address is derived from the current DS register combined with the offset given in the DX register. That is, the DX register in CP/M-86 performs the same function as the DE pair in CP/M-80, with the assumption that DS is properly set. This poses no particular problem for programs which use only a single data segment (as is the case for programs converted from CP/M-80), but when the data group exceeds a single segment, you must ensure that the DS register is set to the segment containing the data area related to the call. It should also be noted that zero values are returned for function calls which are out-of-range.

A list of CP/M-86 calls is given in Table 4-2 with an asterisk following functions which differ from or are added to the set of CP/M-80 Version 2 functions.

Table 4-2. CP/M-86 BDOS Functions

F#	Result	F#	Result
0*	System Reset	24	Return Login Vector
1	Console Input	25	Return Current Disk
2	Console Output	26	Set DMA Address
3	Reader Input	27*	Get Addr(Alloc)
4	Punch Output	28	Write Protect Disk
5	List Output	29	Get Addr(R/O Vector)
6*	Direct Console I/O	30	Set File Attributes
7	Get I/O Byte	31*	Get Addr(Disk Parms)
8	Set I/O Byte	32	Set/Get User Code
9	Print String	33	Read Random
10	Read Console Buffer	34	Write Random
11	Get Console Status	35	Compute File Size
12	Return Version Number	36	Set Random Record
13	Reset Disk System	37*	Reset drive
14	Select Disk	40	Write Random with Zero Fill
15	Open File	50*	Direct BIOS Call
16	Close File	51*	Set DMA Segment Base
17	Search for First	52*	Get DMA Segment Base
18	Search for Next	53*	Get Max Memory Available
19	Delete File	54*	Get Max Mem at Abs Location
20	Read Sequential	55*	Get Memory Region
21	Write Sequential	56*	Get Absolute Memory Region
22	Make File	57*	Free memory region
23	Rename File	58*	Free all memory
		59*	Program load

The individual BDOS functions are described below in three sections which cover the simple functions, file operations, and extended operations for memory management and program loading.

4.2 Simple BDOS Calls

The first set of BDOS functions cover the range 0 through 12, and perform simple functions such as system reset and single character I/O.

Entry		Return
CL: 00H	FUNCTION 0	
DL: Abort Code	SYSTEM RESET	

The system reset function returns control to the CP/M operating system at the CCP command level. The abort code in DL has two possible values: if DL = 00H then the currently active program is terminated and control is returned to the CCP. If DL is a 01H, the program remains in memory and the memory allocation state remains unchanged.

Entry		Return
CL: 01H	FUNCTION 1	AL: ASCII Character
	CONSOLE INPUT	

The console input function reads the next character from the logical console device (CONSOLE) to register AL. Graphic characters, along with carriage return, line feed, and backspace (CONTROL-H) are echoed to the console. Tab characters (CONTROL-I) are expanded in columns of eight characters. The BDOS does not return to the calling program until a character has been typed, thus suspending execution if a character is not ready.

Entry		Return
CL: 02H	FUNCTION 2	
DL: ASCII Character	CONSOLE OUTPUT	

The ASCII character from DL is sent to the logical console. Tab characters expand in columns of eight characters. In addition, a check is made for start/stop scroll (CONTROL-S).

Entry		Return
CL: 03H	FUNCTION 3 READER INPUT	AL: ASCII Character

The Reader Input function reads the next character from the logical reader (READER) into register AL. Control does not return until the character has been read.

Entry		Return
CL: 04H DL: ASCII Character	FUNCTION 4 PUNCH OUTPUT	

The Punch Output function sends the character from register DL to the logical punch device (PUNCH).

Entry		Return
CL: 05H DL: ASCII Character	FUNCTION 5 LIST OUTPUT	

The List Output function sends the ASCII character in register DL to the logical list device (LIST).

```
     Entry                                        Return
 ─────────────►   ┌─────────────────────┐   ─────────────►
 CL: 06H          │ FUNCTION  6         │   AL: char or status
                  │                     │
 DL: 0FFH (input) │ DIRECT CONSOLE I/O  │       (no value)
      or          └─────────────────────┘
     0FEH (status)
      or
     char (output)
```

Direct console I/O is supported under CP/M-86 for those specialized applications where unadorned console input and output is required. Use of this function should, in general, be avoided since it bypasses all of CP/M-86's normal control character functions (e.g., CONTROL-S and CONTROL-P). Programs which perform direct I/O through the BIOS under previous releases of CP/M-80, however, should be changed to use direct I/O under the BDOS so that they can be fully supported under future releases of MP/M and CP/M.

Upon entry to function 6, register DL either contains (1) a hexadecimal FF, denoting a CONSOLE input request, or (2) a hexadecimal FE, denoting a CONSOLE status request, or (3) an ASCII character to be output to CONSOLE where CONSOLE is the logical console device. If the input value is FF, then function 6 directly calls the BIOS console input primitive. The next console input character is returned in AL. If the input value is FE, then function 6 returns AL = 00 if no character is ready and AL = FF otherwise. If the input value in DL is not FE or FF, then function 6 assumes that DL contains a valid ASCII character which is sent to the console.

```
     Entry                                        Return
 ─────────────►   ┌─────────────────────┐   ─────────────►
    CL: 07H       │ FUNCTION  7         │   AL: I/O Byte Value
                  │                     │
                  │ GET I/O BYTE        │
                  └─────────────────────┘
```

The Get I/O Byte function returns the current value of IOBYTE in register AL. The IOBYTE contains the current assignments for the logical devices CONSOLE, READER, PUNCH, and LIST provided the IOBYTE facility is implemented in the BIOS.

Entry		Return
CL: 08H	FUNCTION 8	
DL: I/O Byte Value	SET I/O BYTE	

The Set I/O Byte function changes the system IOBYTE value to that given in register DL. This function allows transient program access to the IOBYTE in order to modify the current assignments for the logical devices CONSOLE, READER, PUNCH, and LIST.

Entry		Return
CL: 09H	FUNCTION 9	
DX: String Offset	PRINT STRING	

The Print String function sends the character string stored in memory at the location given by DX to the logical console device (CONSOLE), until a "$" is encountered in the string. Tabs are expanded as in function 2, and checks are made for start/stop scroll and printer echo.

Entry		Return
CL: 0AH	FUNCTION 10	Console Characters
DX: Buffer Offset	READ CONSOLE BUFFER	in Buffer

The Read Buffer function reads a line of edited console input into a buffer addressed by register DX from the logical console device (CONSOLE). Console input is terminated when either the input buffer is filled or when a return (CONTROL-M) or a line feed (CONTROL-J) character is entered. The input buffer addressed by DX takes the form:

```
DX: +0 +1 +2 +3 +4 +5 +6 +7 +8     . . .    +n
   |mx|nc|c1|c2|c3|c4|c5|c6|c7|    . . .   |??|
```

where "mx" is the maximum number of characters which the buffer will hold, and "nc" is the number of characters placed in the buffer. The characters entered by the operator follow the "nc" value. The value "mx" must be set prior to making a function 10 call and may range in value from 1 to 255. Setting mx to zero is equivalent to setting mx to one. The value "nc" is returned to the user and may range from 0 to mx. If nc < mx, then uninitialized positions follow the last character, denoted by "??" in the above figure. Note that a terminating return or line feed character is not placed in the buffer and not included in the count "nc".

A number of editing control functions are supported during console input under function 10. These are summarized in Table 4-3.

Table 4-3. Line Editing Controls

Keystroke	Result
rub/del	removes and echoes the last character
CONTROL-C	reboots when at the beginning of line
CONTROL-E	causes physical end of line
CONTROL-H	backspaces one character position
CONTROL-J	(line feed) terminates input line
CONTROL-M	(return) terminates input line
CONTROL-R	retypes the current line after new line
CONTROL-U	removes current line after new line
CONTROL-X	backspaces to beginning of current line

Certain functions which return the carriage to the leftmost position (e.g., CONTROL-X) do so only to the column position where the prompt ended. This convention makes operator data input and line correction more legible.

```
        Entry                                 Return
  ------------->  +----------------------+  ------------->
     CL: 0BH      |     FUNCTION 11      |  AL: Console Status
                  | GET CONSOLE STATUS   |
                  +----------------------+
```

The Console Status function checks to see if a character has been typed at the logical console device (CONSOLE). If a character is ready, the value 01H is returned in register AL. Otherwise a 00H value is returned.

```
        Entry                                 Return
  ------------->  +-----------------------+  ------------->
     CL: 0CH      |      FUNCTION 12      |  BX: Version Number
                  | RETURN VERSION NUMBER |
                  +-----------------------+
```

Function 12 provides information which allows version independent programming. A two-byte value is returned, with BH = 00 designating the CP/M release (BH = 01 for MP/M), and BL = 00 for all releases previous to 2.0. CP/M 2.0 returns a hexadecimal 20 in register BL, with subsequent version 2 releases in the hexadecimal range 21, 22, through 2F. To provide version number compatibility, the initial release of CP/M-86 returns a 2.2.

4.3 BDOS File Operations

Functions 12 through 52 are related to disk file operations under CP/M-86. In many of these operations, DX provides the DS-relative offset to a file control block (FCB). The File Control Block (FCB) data area consists of a sequence of 33 bytes for sequential access, or a sequence of 36 bytes in the case that the file is accessed randomly. The default file control block normally located at offset 005CH from the DS register can be used for random access files, since bytes 007DH, 007EH, and 007FH are available for this purpose. Here is the FCB format, followed by definitions of each of its fields:

dr	f1	f2	/ /	f8	t1	t2	t3	ex	s1	s2	rc	d0	/ /	dn	cr	r0	r1	r2
00	01	02	...	08	09	10	11	12	13	14	15	16	...	31	32	33	34	35

where

dr drive code (0 - 16)
0 => use default drive for file
1 => auto disk select drive A,
2 => auto disk select drive B,
...
16=> auto disk select drive P.

f1...f8 contain the file name in ASCII upper case, with high bit = 0

t1,t2,t3 contain the file type in ASCII upper case, with high bit = 0
t1', t2', and t3' denote the high bit of these positions,
t1' = 1 => Read/Only file,
t2' = 1 => SYS file, no DIR list

ex contains the current extent number, normally set to 00 by the user, but in range 0 - 31 during file I/O

s1 reserved for internal system use

s2 reserved for internal system use, set to zero on call to OPEN, MAKE, SEARCH

rc record count for extent "ex," takes on values from 0 - 128

d0...dn filled-in by CP/M, reserved for system use

cr current record to read or write in a sequential file operation, normally set to zero by user

r0,r1,r2 optional random record number in the range 0-65535, with overflow to r2, r0,r1 constitute a 16-bit value with low byte r0, and high byte r1

For users of earlier versions of CP/M, it should be noted in passing that both CP/M Version 2 and CP/M-86 perform directory operations in a reserved area of memory that does not affect write buffer content, except in the case of Search and Search Next where the directory record is copied to the current DMA address.

There are three error situations that the BDOS may encounter during file processing, initiated as a result of a BDOS File I/O function call. When one of these conditions is detected, the BDOS issues the following message to the console:

```
BDOS ERR ON x: error
```

where x is the drive name of the drive selected when the error condition is detected, and "error" is one of the three messages:

```
BAD SECTOR      SELECT      R/O
```

These error situations are trapped by the BDOS, and thus the executing transient program is temporarily halted when the error is detected. No indication of the error situation is returned to the transient program.

The "BAD SECTOR" error is issued as the result of an error condition returned to the BDOS from the BIOS module. The BDOS makes BIOS sector read and write commands as part of the execution of BDOS file related system calls. If the BIOS read or write routine detects a hardware error, it returns an error code to the BDOS resulting in this error message. The operator may respond to this error in two ways: a CONTROL-C terminates the executing program, while a RETURN instructs CP/M-86 to ignore the error and allow the program to continue execution.

The "SELECT" error is also issued as the result of an error condition returned to the BDOS from the BIOS module. The BDOS makes a BIOS disk select call prior to issuing any BIOS read or write to a particular drive. If the selected drive is not supported in the BIOS module, it returns an error code to the BDOS resulting in this error message. CP/M-86 terminates the currently running program and returns to the command level of the CCP following any input from the console.

The "R/O" message occurs when the BDOS receives a command to write to a drive that is in read-only status. Drives may be placed in read-only status explicitly as the result of a STAT command or BDOS function call, or implicitly if the BDOS detects that disk media has been changed without performing a "warm start." The ability to detect changed media is optionally included in the BIOS, and exists only if a checksum vector is included for the selected drive. Upon entry of any character at the keyboard, the transient program is aborted, and control returns to the CCP.

Entry		Return
CL: 0DH	FUNCTION 13 RESET DISK SYSTEM	

The Reset Disk Function is used to programmatically restore the file system to a reset state where all disks are set to read/write (see functions 28 and 29), only disk drive A is selected. This function can be used, for example, by an application program which requires disk changes during operation. Function 37 (Reset Drive) can also be used for this purpose.

Entry		Return
CL: 0EH DL: Selected Disk	FUNCTION 14 SELECT DISK	

The Select Disk function designates the disk drive named in register DL as the default disk for subsequent file operations, with DL = 0 for drive A, 1 for drive B, and so-forth through 15 corresponding to drive P in a full sixteen drive system. In addition, the designated drive is logged-in if it is currently in the reset state. Logging-in a drive places it in "on-line" status which activates the drive's directory until the next cold start, warm start, disk system reset, or drive reset operation. FCB's which specify drive code zero (dr = 00H) automatically reference the currently selected default drive. Drive code values between 1 and 16, however, ignore the selected default drive and directly reference drives A through P.

Entry		Return
CL: 0FH DX: FCB Offset	FUNCTION 15 OPEN FILE	AL: Return Code

The Open File operation is used to activate a FCB specifying a file which currently exists in the disk directory for the currently active user number. The BDOS scans the disk directory of the drive specified by byte 0 of the FCB referenced by DX for a match in positions 1 through 12 of the referenced FCB, where an ASCII question mark (3FH) matches any directory character in any of these positions. Normally, no question marks are included and, further, byte "ex" of the FCB is set to zero before making the open call.

If a directory element is matched, the relevant directory information is copied into bytes d0 through dn of the FCB, thus allowing access to the files through subsequent read and write operations. Note that an existing file must not be accessed until a successful open operation is completed. Further, an FCB not activated by either an open or make function must not be used in BDOS read or write commands. Upon return, the open function returns a "directory code" with the value 0 through 3 if the open was successful, or 0FFH (255 decimal) if the file cannot be found. If question marks occur in the FCB then the first matching FCB is activated. Note that the current record ("cr") must be zeroed by the program if the file is to be accessed sequentially from the first record.

Entry		Return
CL: 10H	FUNCTION 16	AL: Return Code
DX: FCB Offset	CLOSE FILE	

The Close File function performs the inverse of the open file function. Given that the FCB addressed by DX has been previously activated through an open or make function (see functions 15 and 22), the close function permanently records the new FCB in the referenced disk directory. The FCB matching process for the close is identical to the open function. The directory code returned for a successful close operation is 0, 1, 2, or 3, while a 0FFH (255 decimal) is returned if the file name cannot be found in the directory. A file need not be closed if only read operations have taken place. If write operations have occurred, however, the close operation is necessary to permanently record the new directory information.

Entry		Return
CL: 11H	FUNCTION 17	AL: Directory Code
DX: FCB Offset	SEARCH FOR FIRST	

Search First scans the directory for a match with the file given by the FCB addressed by DX. The value 255 (hexadecimal FF) is returned if the file is not found, otherwise 0, 1, 2, or 3 is returned indicating the file is present. In the case that the file is found, the buffer at the current DMA address is filled with the record containing the directory entry, and its relative starting position is AL * 32 (i.e., rotate the AL register left 5 bits). Although not normally required for application programs, the directory information can be extracted from the buffer at this position.

An ASCII question mark (63 decimal, 3F hexadecimal) in any position from "f1" through "ex" matches the corresponding field of any directory entry on the default or auto-selected disk drive. If the "dr" field contains an ASCII question mark, then the auto disk select function is disabled, the default disk is searched, with the search function returning any matched entry, allocated or free, belonging to any user number. This latter function is not normally used by application programs, but does allow complete flexibility to scan all current directory values. If the "dr" field is not a question mark, the "s2" byte is automatically zeroed.

Entry		Return
CL: 12H	FUNCTION 18	AL: Directory Code
	SEARCH FOR NEXT	

The Search Next function is similar to the Search First function, except that the directory scan continues from the last matched entry. Similar to function 17, function 18 returns the decimal value 255 in A when no more directory items match. In terms of execution sequence, a function 18 call must follow either a function 17 or function 18 call with no other intervening BDOS disk related function calls.

Entry		Return
CL: 13H	FUNCTION 19	AL: Return Code
DX: FCB Offset	DELETE FILE	

The Delete File function removes files which match the FCB addressed by DX. The filename and type may contain ambiguous references (i.e., question marks in various positions), but the drive select code cannot be ambiguous, as in the Search and Search Next functions. Function 19 returns a 0FFH (decimal 255) if the referenced file or files cannot be found, otherwise a value of zero is returned.

Entry		Return
CL: 14H	FUNCTION 20	AL: Return Code
DX: FCB Offset	READ SEQUENTIAL	

Given that the FCB addressed by DX has been activated through an open or make function (numbers 15 and 22), the Read Sequential function reads the next 128 byte record from the file into memory at the current DMA address. The record is read from position "cr" of the extent, and the "cr" field is automatically incremented to the next record position. If the "cr" field overflows then the next logical extent is automatically opened and the "cr" field is reset to zero in preparation for the next read operation. The "cr" field must be set to zero following the open call by the user if the intent is to read sequentially from the beginning of the file. The value 00H is returned in the AL register if the read operation was successful, while a value of 01H is returned if no data exists at the next record position of the file. Normally, the no data situation is encountered at the end of a file. However, it can also occur if an attempt is made to read a data block which has not been previously written, or an extent which has not been created. These situations are usually restricted to files created or appended by use of the BDOS Write Random commmand (function 34).

Entry		Return
CL: 15H	FUNCTION 21	AL: Return Code
DX: FCB Offset	WRITE SEQUENTIAL	

Given that the FCB addressed by DX has been activated through an open or make function (numbers 15 and 22), the Write Sequential function writes the 128 byte data record at the current DMA address to the file named by the FCB. The record is placed at position "cr" of the file, and the "cr" field is automatically incremented to the next record position. If the "cr" field overflows then the next logical extent is automatically opened and the "cr" field is reset to zero in preparation for the next write operation. Write operations can take place into an existing file, in which case newly written records overlay those which already exist in the file. The "cr" field must be set to zero following an open or make call by the user if the intent is to write sequentially from the beginning of the file. Register AL = 00H upon return from a successful write operation, while a non-zero value indicates an unsuccessful write due to one of the following conditions:

01 No available directory space - This condition occurs when the write command attempts to create a new extent that requires a new directory entry and no available directory entries exist on the selected disk drive.

02 No available data block - This condition is encountered when the write command attempts to allocate a new data block to the file and no unallocated data blocks exist on the selected disk drive.

Entry		Return
CL: 16H	FUNCTION 22	AL: Return Code
DX: FCB Offset	MAKE FILE	

The Make File operation is similar to the open file operation except that the FCB must name a file which does not exist in the currently referenced disk directory (i.e., the one named explicitly by a non-zero "dr" code, or the default disk if "dr" is zero). The BDOS creates the file and initializes both the directory and main memory value to an empty file. The programmer must ensure that no duplicate file names occur, and a preceding delete operation is sufficient if there is any possibility of duplication. Upon return, register A = 0, 1, 2, or 3 if the operation was successful and 0FFH (255 decimal) if no more directory space is available. The make function has the side-effect of activating the FCB and thus a subsequent open is not necessary.

Entry		Return
CL: 17H	FUNCTION 23	AL: Return Code
DX: FCB Offset	RENAME FILE	

The Rename function uses the FCB addressed by DX to change all directory entries of the file specified by the file name in the first 16 bytes of the FCB to the file name in the second 16 bytes. It is the user's responsibility to insure that the file names specified are valid CP/M unambiguous file names. The drive code "dr" at position 0 is used to select the drive, while the drive code for the new file name at position 16 of the FCB is ignored. Upon return, register AL is set to a value of zero if the rename was successful, and 0FFH (255 decimal) if the first file name could not be found in the directory scan.

Entry		Return
CL: 18H	FUNCTION 24	BX: Login Vector
BX: Login Vector	RETURN LOGIN VECTOR	

The login vector value returned by CP/M-86 is a 16-bit value in BX, where the least significant bit corresponds to the first drive A, and the high order bit corresponds to the sixteenth drive, labelled P. A "0" bit indicates that the drive is not on-line, while a "1" bit marks an drive that is actively on-line due to an explicit disk drive selection, or an implicit drive select caused by a file operation which specified a non-zero "dr" field.

Entry		Return
CL: 19H	FUNCTION 25	AL: Current Disk
	RETURN CURRENT DISK	

Function 25 returns the currently selected default disk number in register AL. The disk numbers range from 0 through 15 corresponding to drives A through P.

Entry		Return
CL: 1AH	FUNCTION 26	
DX: DMA Offset	SET DMA ADDRESS	

"DMA" is an acronym for Direct Memory Address, which is often used in connection with disk controllers which directly access the memory of the mainframe computer to transfer data to and from the disk subsystem. Although many computer systems use non-DMA access (i.e., the data is transfered through programmed I/O operations), the DMA address has, in CP/M, come to mean the address at which the 128 byte data record resides before a disk write and after a disk read. In the CP/M-86 environment, the Set DMA function is used to specify the offset of the read or write buffer from the current DMA base. Therefore, to specify the DMA address, both a function 26 call and a function 51 call are required. Thus, the DMA address becomes the value specified by DX plus the DMA base value until it is changed by a subsequent Set DMA or set DMA base function.

Entry		Return
CL: 1BH	FUNCTION 27	BX: ALLOC Offset
	GET ADDR(ALLOC)	ES: Segment base

An "allocation vector" is maintained in main memory for each on-line disk drive. Various system programs use the information provided by the allocation vector to determine the amount of remaining storage (see the STAT program). Function 27 returns the segment base and the offset address of the allocation vector for the currently selected disk drive. The allocation information may, however, be invalid if the selected disk has been marked read/only.

Entry		Return
CL: 1CH	FUNCTION 28	
	WRITE PROTECT DISK	

The disk write protect function provides temporary write protection for the currently selected disk. Any attempt to write to the disk, before the next cold start, warm start, disk system reset, or drive reset operation produces the message:

```
Bdos Err on d: R/O
```

Entry		Return
CL: 1DH	FUNCTION 29 GET READ/ONLY VECTOR	BX: R/O Vector Value

Function 29 returns a bit vector in register BX which indicates drives which have the temporary read/only bit set. Similar to function 24, the least significant bit corresponds to drive A, while the most significant bit corresponds to drive P. The R/O bit is set either by an explicit call to function 28, or by the automatic software mechanisms within CP/M-86 which detect changed disks.

Entry		Return
CL: 1EH DX: FCB Offset	FUNCTION 30 SET FILE ATTRIBUTES	AL: Return Code

The Set File Attributes function allows programmatic manipulation of permanent indicators attached to files. In particular, the R/O, System and Archive attributes (t1', t2', and t3') can be set or reset. The DX pair addresses a FCB containing a file name with the appropriate attributes set or reset. It is the user's responsibility to insure that an ambiguous file name is not specified. Function 30 searches the default disk drive directory area for directory entries that belong to the current user number and that match the FCB specified name and type fields. All matching directory entries are updated to contain the selected indicators. Indicators f1' through f4' are not presently used, but may be useful for applications programs, since they are not involved in the matching process during file open and close operations. Indicators f5' through f8' are reserved for future system expansion. The currently assigned attributes are defined as follows:

t1': The R/O attribute indicates if set that the file is in read/only status. BDOS will not allow write commands to be issued to files in R/O status.

t2': The System attribute is referenced by the CP/M DIR utility. If set, DIR will not display the file in a directory display.

t3': The Archive attribute is reserved but not actually used by CP/M-86 If set it indicates that the file has been written to back up storage by a user written archive program. To implement this facility, the archive program sets this attribute when it copies a file to back up storage; any programs updating or creating files reset this attribute. Further, the archive program backs up only those files that have the Archive attribute reset. Thus, an automatic back up facility restricted to modified files can be easily implemented.

Function 30 returns with register AL set to 0FFH (255 decimal) if the referenced file cannot be found, otherwise a value of zero is returned.

Entry		Return
CL: 1FH	FUNCTION 31	BX: DPB Offset
	GET ADDR (DISK PARMS)	ES: Segment Base

The offset and the segment base of the BIOS resident disk parameter block of the currently selected drive are returned in BX and ES as a result of this function call. This control block can be used for either of two purposes. First, the disk parameter values can be extracted for display and space computation purposes, or transient programs can dynamically change the values of current disk parameters when the disk environment changes, if required. Normally, application programs will not require this facility. Section 6.3 defines the BIOS disk parameter block.

Entry		Return
CL: 20H	FUNCTION 32	AL: Current Code or no value
DL: 0FFH(get) or User Code (set)	SET/GET USER CODE	

An application program can change or interrogate the currently active user number by calling function 32. If register DL = 0FFH, then the value of the current user number is returned in register AL, where the value is in the range 0 to 15. If register DL is not 0FFH, then the current user number is changed to the value of DL (modulo 16).

Entry		Return
CL: 21H	FUNCTION 33	AL: Return Code
DX: FCB Offset	READ RANDOM	

The Read Random function is similar to the sequential file read operation of previous releases, except that the read operation takes place at a particular record number, selected by the 24-bit value constructed from the three byte field following the FCB (byte positions r0 at 33, r1 at 34, and r2 at 35). Note that the sequence of 24 bits is stored with least significant byte first (r0), middle byte next (r1), and high byte last (r2). CP/M does not reference byte r2, except in computing the size of a file (function 35). Byte r2 must be zero, however, since a non-zero value indicates overflow past the end of file.

Thus, the r0,r1 byte pair is treated as a double-byte, or "word" value, which contains the record to read. This value ranges from 0 to 65535, providing access to any particular record of any size file. In order to access a file using the Read Random function, the base extent (extent 0) must first be opened. Although the base extent may or may not contain any allocated data, this ensures that the FCB is properly initialized for subsequent random access operations. The selected record number is then stored into the random record field (r0,r1), and the BDOS is called to read the record. Upon return from the call, register AL either contains an error code, as listed below, or the value 00 indicating the operation was successful. In the latter case, the buffer at the current DMA address contains the randomly accessed record. Note that contrary to the sequential read operation, the record number is not advanced. Thus, subsequent random read operations continue to read the same record.

Upon each random read operation, the logical extent and current record values are automatically set. Thus, the file can be sequentially read or written, starting from the current randomly accessed position. Note, however, that in this case, the last randomly read record will be re-read as you switch from random mode to sequential read, and the last record will be re-written as you switch to a sequential write operation. You can, of course, simply advance the random record position following each random read or write to obtain the effect of a sequential I/O operation.

Error codes returned in register AL following a random read are listed in Table 4-4, below.

Table 4-4. Function 33 (Read Random) Error Codes

Code	Meaning
01	Reading unwritten data - This error code is returned when a random read operation accesses a data block which has not been previously written.
02	(not returned by the Random Read command)
03	Cannot close current extent - This error code is returned when BDOS cannot close the current extent prior to moving to the new extent containing the record specified by bytes r0,r1 of the FCB. This error can be caused by an overwritten FCB or a read random operation on an FCB that has not been opened.
04	Seek to unwritten extent - This error code is returned when a random read operation accesses an extent that has not been created. This error situation is equivalent to error 01.
05	(not returned by the Random Read command)
06	Random record number out of range - This error code is returned whenever byte r2 of the FCB is non-zero.

Normally, non-zero return codes can be treated as missing data, with zero return codes indicating operation complete.

Entry		Return
CL: 22H	FUNCTION 34	AL: Return Code
DX: FCB Offset	WRITE RANDOM	

The Write Random operation is initiated similar to the Read Random call, except that data is written to the disk from the current DMA address. Further, if the disk extent or data block which is the target of the write has not yet been allocated, the allocation is performed before the write operation continues. As in the Read Random operation, the random record number is not changed as a result of the write. The logical extent number and current record positions of the file control block are set to correspond to the random record which is being written. Sequential read or write operations can commence following a random write, with the note that the currently addressed record is either read or rewritten again as the sequential operation begins. You can also simply advance the random record position following each write to get the effect of a sequential write operation. In particular, reading or writing the last record of an extent in random mode does not cause an automatic extent switch as it does in sequential mode.

In order to access a file using the Write Random function, the base extent (extent 0) must first be opened. As in the Read Random function, this ensures that the FCB is properly initialized for subsequent random access operations. If the file is empty, a Make File function must be issued for the base extent. Although the base extent may or may not contain any allocated data, this ensures that the file is properly recorded in the directory, and is visible in DIR requests.

Upon return from a Write Random call, register AL either contains an error code, as listed in Table 4-5 below, or the value 00 indicating the operation was successful.

Table 4-5. Function 34 (WRITE RANDOM) Error Codes

Code	Meaning
01	(not returned by the Random Write command)
02	No available data block - This condition is encountered when the Write Random command attempts to allocate a new data block to the file and no unallocated data blocks exist on the selected disk drive.

Table 4-5. (continued)

Code	Meaning
03	Cannot close current extent - This error code is returned when BDOS cannot close the current extent prior to moving to the new extent containing the record specified by bytes r0,r1 of the FCB. This error can be caused by an overwritten FCB or a write random operation on an FCB that has not been opened.
04	(not returned by the Random Write command)
05	No available directory space - This condition occurs when the write command attempts to create a new extent that requires a new directory entry and no available directory entries exist on the selected disk drive.
06	Random record number out of range - This error code is returned whenever byte r2 of the FCB is non-zero.

Entry		Return
CL: 23H	FUNCTION 35	Random Record Field Set
DX: FCB Offset	COMPUTE FILE SIZE	

When computing the size of a file, the DX register addresses an FCB in random mode format (bytes r0, r1, and r2 are present). The FCB contains an unambiguous file name which is used in the directory scan. Upon return, the random record bytes contain the "virtual" file size which is, in effect, the record address of the record following the end of the file. If, following a call to function 35, the high record byte r2 is 01, then the file contains the maximum record count 65536. Otherwise, bytes r0 and r1 constitute a 16-bit value (r0 is the least significant byte, as before) which is the file size.

Data can be appended to the end of an existing file by simply calling function 35 to set the random record position to the end of file, then performing a sequence of random writes starting at the preset record address.

The virtual size of a file corresponds to the physical size when the file is written sequentially. If, instead, the file was created in random mode and "holes" exist in the allocation, then the file may in fact contain fewer records than the size indicates. If, for example, a single record with record number 65535 (CP/M's maximum record number) is written to a file using the Write Random function, then the virtual size of the file is 65536 records, although only one block of data is actually allocated.

Entry		Return
CL: 24H	FUNCTION 36	Random Record Field Set
DX: FCB Offset	SET RANDOM RECORD	

The Set Random Record function causes the BDOS to automatically produce the random record position of the next record to be accessed from a file which has been read or written sequentially to a particular point. The function can be useful in two ways.

First, it is often necessary to initially read and scan a sequential file to extract the positions of various "key" fields. As each key is encountered, function 36 is called to compute the random record position for the data corresponding to this key. If the data unit size is 128 bytes, the resulting record position minus one is placed into a table with the key for later retrieval. After scanning the entire file and tabularizing the keys and their record numbers, you can move instantly to a particular keyed record by performing a random read using the corresponding random record number which was saved earlier. The scheme is easily generalized when variable record lengths are involved since the program need only store the buffer-relative byte position along with the key and record number in order to find the exact starting position of the keyed data at a later time.

A second use of function 36 occurs when switching from a sequential read or write over to random read or write. A file is sequentially accessed to a particular point in the file, function 36 is called which sets the record number, and subsequent random read and write operations continue from the next record in the file.

Entry		Return
CL: 25H	FUNCTION 37	AL: 00H
DX: Drive Vector	RESET DRIVE	

The Reset Drive function is used to programmatically restore specified drives to the reset state (a reset drive is not logged-in and is in read/write status). The passed parameter in register DX is a 16 bit vector of drives to be reset, where the least significant bit corresponds to the first drive, A, and the high order bit corresponds to the sixteenth drive, labelled P. Bit values of "1" indicate that the specified drive is to be reset.

In order to maintain compatibility with MP/M, CP/M returns a zero value for this function.

Entry		Return
CL: 28H	FUNCTION 40	AL: Return Code
DX: FCB Offset	WRITE RANDOM WITH ZERO FILL	

The Write Random With Zero Fill function is similar to the Write Random function (function 34) with the exception that a previously unallocated data block is initialized to records filled with zeros before the record is written. If this function has been used to create a file, records accessed by a read random operation that contain all zeros identify unwritten random record numbers. Unwritten random records in allocated data blocks of files created using the Write Random function contain uninitialized data.

Entry		Return
CL: 32H	FUNCTION 50	
DX: BIOS Descriptor	DIRECT BIOS CALL	

Function 50 provides a direct BIOS call and transfers control through the BDOS to the BIOS. The DX register addresses a five-byte memory area containing the BIOS call parameters:

8-bit	16-bit	16-bit
Func	value(CX)	value(DX)

where Func is a BIOS function number, (see Table 5-1), and value(CX) and value(DX) are the 16-bit values which would normally be passed directly in the CX and DX registers with the BIOS call. The CX and DX values are loaded into the 8086 registers before the BIOS call is initiated.

Entry		Return
CL: 33H	FUNCTION 51	
DX: Base Address	SET DMA BASE	

Function 51 sets the base register for subsequent DMA transfers. The word parameter in DX is a paragraph address and is used with the DMA offset to specify the address of a 128 byte buffer area to be used in the disk read and write functions. Note that upon initial program loading, the default DMA base is set to the address of the user's data segment (the initial value of DS) and the DMA offset is set to 0080H, which provides access to the default buffer in the base page.

Entry		Return
CL: 34H	FUNCTION 52	BX: DMA Offset
	GET DMA BASE	ES: DMA Segment

Function 52 returns the current DMA Base Segment address in ES, with the current DMA Offset in DX.

4.4 BDOS Memory Management and Load

Memory is allocated in two distinct ways under CP/M-86. The first is through a static allocation map, located within the BIOS, that defines the physical memory which is available on the host system. In this way, it is possible to operate CP/M-86 in a memory configuration which is a mixture of up to eight non-contiguous areas of RAM or ROM, along with reserved, missing, or faulty memory regions. In a simple RAM-based system with contiguous memory, the static map defines a single region, usually starting at the end of the BIOS and extending up to the end of available memory.

Once memory is physically mapped in this manner, CP/M-86 performs the second level of dynamic allocation to support transient program loading and execution. CP/M-86 allows dynamic allocation of memory into, again, eight regions. A request for allocation takes place either implicitly, through a program load operation, or explicitly through the BDOS calls given in this section. Programs themselves are loaded in two ways: through a command entered at the CCP level, or through the BDOS Program Load operation (function 59). Multiple programs can be loaded at the CCP level, as long as each program executes a System Reset (function 0) and remains in memory (DL = 01H). Multiple programs of this type only receive control by intercepting interrupts, and thus under normal circumstances there is only one transient program in memory at any given time. If, however, multiple programs are present in memory, then CONTROL-C characters entered by the operator delete these programs in the opposite order in which they were loaded no matter which program is actively reading the console.

Any given program loaded through a CCP command can, itself, load additional programs and allocate data areas. Suppose four regions of memory are allocated in the following order: a program is loaded at the CCP level through an operator command. The CMD file header is read, and the entire memory image consisting of the program and its data is loaded into region A, and execution begins. This program, in turn, calls the BDOS Program Load function (59) to load another program into region B, and transfers control to the loaded program. The region B program then allocates an additional region C, followed by a region D. The order of allocation is shown in Figure 4-1 below:

Region A
Region B
Region C
Region D

Figure 4-1. Example Memory Allocation

There is a hierarchical ownership of these regions: the program in A controls all memory from A through D. The program in B also controls regions B through D. The program in A can release regions B through D, if desired, and reload yet another program. DDT-86, for example, operates in this manner by executing the Free Memory call (function 57) to release the memory used by the current program before loading another test program. Further, the program in B can release regions C and D if required by the application. It must be noted, however, that if either A or B terminates by a System Reset (BDOS function 0 with DL = 00H) then all four regions A through D are released.

A transient program may release a portion of a region, allowing the released portion to be assigned on the next allocation request. The released portion must, however, be at the beginning or end of the region. Suppose, for example, the program in region B above receives 800H paragraphs at paragraph location 100H following its first allocation request as shown in Figure 4-2 below.

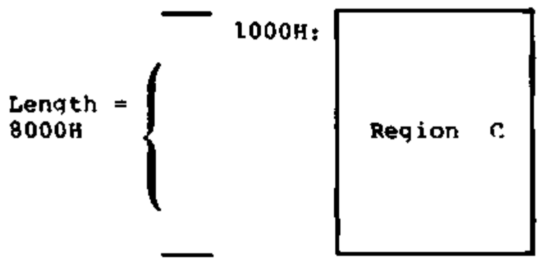

Figure 4-2. Example Memory Region

Suppose further that region D is then allocated. The last 200H paragraphs in region C can be returned without affecting region D by releasing the 200H paragraphs beginning at paragraph base 700H, resulting in the memory arrangement shown in Figure 4-3.

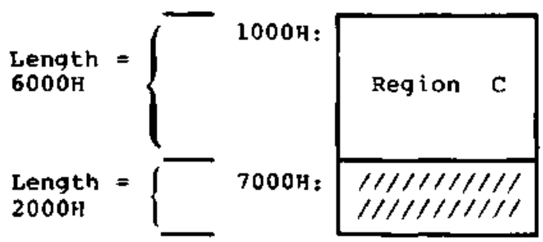

Figure 4-3. Example Memory Regions

The region beginning at paragraph address 700H is now available for allocation in the next request. Note that a memory request will fail if eight memory regions have already been allocated. Normally, if all program units can reside in a contiguous region, the system allocates only one region.

Memory management functions beginning at 53 reference a Memory Control Block (MCB), defined in the calling program, which takes the form:

	16-bit	16-bit	8-bit
MCB:	M-Base	M-Length	M-Ext

where M-Base and M-Length are either input or output values expressed in 16-byte paragraph units, and M-Ext is a returned byte value, as defined specifically with each function code. An error condition is normally flagged with a 0FFH returned value in order to match the file error conventions of CP/M.

Entry		Return
CL: 35H	FUNCTION 53	AL: Return Code
DX: Offset of MCB	GET MAX MEM	

Function 53 finds the largest available memory region which is less than or equal to M-Length paragraphs. If successful, M-Base is set to the base paragraph address of the available area, and M-Length to the paragraph length. AL has the value 0FFH upon return if no memory is available, and 00H if the request was successful. M-Ext is set to 1 if there is additional memory for allocation, and 0 if no additional memory is available.

Entry		Return
CL: 36H	FUNCTION 54	AL: Return Code
DX: Offset of MCB	GET ABS MAX	

Function 54 is used to find the largest possible region at the absolute paragraph boundary given by M-Base, for a maximum of M-Length paragraphs. M-Length is set to the actual length if successful. AL has the value 0FFH upon return if no memory is available at the absolute address, and 00H if the request was successful.

```
   Entry                                      Return
 ──────────►  ┌──────────────────┐         ──────────►
 CL: 37H      │   FUNCTION 55    │         AL: Return Code
 DX: Offset   │    ALLOC MEM     │
     of MCB   └──────────────────┘
```

The allocate memory function allocates a memory area according to the MCB addressed by DX. The allocation request size is obtained from M-Length. Function 55 returns in the user's MCB the base paragraph address of the allocated region. Register AL contains a 00H if the request was successful and a 0FFH if the memory could not be allocated.

```
   Entry                                      Return
 ──────────►  ┌──────────────────┐         ──────────►
 CL: 38H      │   FUNCTION 56    │         AL: Return Code
 DX: Offset   │  ALLOC ABS MEM   │
     of MCB   └──────────────────┘
```

The allocate absolute memory function allocates a memory area according to the MCB addressed by DX. The allocation request size is obtained from M-Length and the absolute base address from M-Base. Register AL contains a 00H if the request was successful and a 0FFH if the memory could not be allocated.

```
   Entry                                      Return
 ──────────►  ┌──────────────────┐         ──────────►
 CL: 39H      │   FUNCTION 57    │
 DX: Offset   │    FREE MEM      │
     of MCB   └──────────────────┘
```

Function 57 is used to release memory areas allocated to the program. The value of the M-Ext field controls the operation of this function: if M-Ext = 0FFH then all memory areas allocated by the calling program are released. Otherwise, the memory area of length M-Length at location M-Base given in the MCB addressed by DX is released (the M-Ext field should be set to 00H in this case). As described above, either an entire allocated region must be released, or the end of a region must be released: the middle section cannot be returned under CP/M-86.

Entry		Return
CL: 3AH	FUNCTION 58 FREE ALL MEM	

Function 58 is used to release all memory in the CP/M-86 environment (normally used only by the CCP upon initialization).

Entry		Return
CL: 3BH DX: Offset of FCB	FUNCTION 59 PROGRAM LOAD	AX: Return Code/ Base Page Addr BX: Base Page Addr

Function 59 loads a CMD file. Upon entry, register DX contains the DS relative offset of a successfully opened FCB which names the input CMD file. AX has the value 0FFFFH if the program load was unsuccessful. Otherwise, AX and BX both contain the paragraph address of the base page belonging to the loaded program. The base address and segment length of each segment is stored in the base page. Note that upon program load at the CCP level, the DMA base address is initialized to the base page of the loaded program, and the DMA offset address is initialized to 0080H. However, this is a function of the CCP, and a function 59 does not establish a default DMA address. It is the responsibility of the program which executes function 59 to execute function 51 to set the DMA base and function 26 to set the DMA offset before passing control to the loaded program.

Section 5
Basic I/O System (BIOS) Organization

The distribution version of CP/M-86 is setup for operation with the Intel SBC 86/12 microcomputer and an Intel 204 diskette controller. All hardware dependencies are, however, concentrated in subroutines which are collectively referred to as the Basic I/O System, or BIOS. A CP/M-86 system implementor can modify these subroutines, as described below, to tailor CP/M-86 to fit nearly any 8086 or 8088 operating environment. This section describes the actions of each BIOS entry point, and defines variables and tables referenced within the BIOS. The discussion of Disk Definition Tables is, however, treated separately in the next section of this manual.

5.1 Organization of the BIOS

The BIOS portion of CP/M-86 resides in the topmost portion of the operating system (highest addresses), and takes the general form shown in Figure 5-1, below:

CS, DS, ES, SS:	Console Command Processor and Basic Disk Operating System
CS + 2500H:	BIOS Jump Vector
CS + 253FH:	BIOS Entry Points
BIOS:	Disk Parameter Tables
	Uninitialized Scratch RAM

Figure 5-1. General CP/M-86 Organization

As described in the following sections, the CCP and BDOS are supplied with CP/M-86 in hex file form as CPM.H86. In order to implement CP/M-86 on non-standard hardware, you must create a BIOS which performs the functions listed below and concatenate the resulting hex file to the end of the CPM.H86 file. The GENCMD utility is then used to produce the CPM.SYS file for subsequent load by the cold start loader. The cold start loader that loads the CPM.SYS file into memory contains a simplified form of the BIOS, called the LDBIOS (Loader BIOS). It loads CPM.SYS into memory at the location defined in the CPM.SYS header (usually 0400H). The procedure to follow in construction and execution of the cold start loader and the CP/M-86 Loader is given in a later section.

Appendix D contains a listing of the standard CP/M-86 BIOS for the Intel SBC 86/12 system using the Intel 204 Controller Board. Appendix E shows a sample "skeletal" BIOS called CBIOS that contains the essential elements with the device drivers removed. You may wish to review these listings in order to determine the overall structure of the BIOS.

5.2 The BIOS Jump Vector

Entry to the BIOS is through a "jump vector" located at offset 2500H from the base of the operating system. The jump vector is a sequence of 21 three-byte jump instructions which transfer program control to the individual BIOS entry points. Although some non-essential BIOS subroutines may contain a single return (RET) instruction, the corresponding jump vector element must be present in the order shown below in Table 5-1. An example of a BIOS jump vector may be found in Appendix D, in the standard CP/M-86 BIOS listing.

Parameters for the individual subroutines in the BIOS are passed in the CX and DX registers, when required. CX receives the first parameter; DX is used for a second argument. Return values are passed in the registers acco ding to type: Byte values are returned in AL. Word values (16 bits) are returned in BX. Specific parameters and returned values are described with each subroutine.

Table 5-1. BIOS Jump Vector

Offset from Beginning of BIOS	Suggested Instruction	BIOS F#	Description
2500H	JMP INIT	0	Arrive Here from Cold Boot
2503H	JMP WBOOT	1	Arrive Here for Warm Start
2506H	JMP CONST	2	Check for Console Char Ready
2509H	JMP CONIN	3	Read Console Character In
250CH	JMP CONOUT	4	Write Console Character Out
250FH	JMP LIST	5	Write Listing Character Out
2512H	JMP PUNCH	6	Write Char to Punch Device
2515H	JMP READER	7	Read Reader Device
2518H	JMP HOME	8	Move to Track 00
251BH	JMP SELDSK	9	Select Disk Drive
251EH	JMP SETTRK	10	Set Track Number
2521H	JMP SETSEC	11	Set Sector Number
2524H	JMP SETDMA	12	Set DMA Offset Address
2527H	JMP READ	13	Read Selected Sector
252AH	JMP WRITE	14	Write Selected Sector
252DH	JMP LISTST	15	Return List Status
2530H	JMP SECTRAN	16	Sector Translate
2533H	JMP SETDMAB	17	Set DMA Segment Address
2536H	JMP GETSEGB	18	Get MEM DESC Table Offset
2539H	JMP GETIOB	19	Get I/O Mapping Byte
253CH	JMP SETIOB	20	Set I/O Mapping Byte

There are three major divisions in the BIOS jump table: system (re)initialization subroutines, simple character I/O subroutines, and disk I/O subroutines.

5.3 Simple Peripheral Devices

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-86 as "logical" devices, and are assigned to physical devices within the BIOS. Device characteristics are defined in Table 5-2.

Table 5-2. CP/M-86 Logical Device Characteristics

Device Name	Characteristics
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, your CBIOS should give an appropriate error message so that the system does not "hang" if the device is accessed by PIP or some other transient program. Alternately, the PUNCH and LIST subroutines can just simply return, and the READER subroutine can return with a 1AH (ctl-Z) in reg A to indicate immediate end-of-file.

For added flexibility, you 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-86 processing (see the STAT command). The definition of the IOBYTE function corresponds to the Intel standard as follows: a single location in the BIOS 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	LIST	PUNCH	READER	CONSOLE
	bits 6,7	bits 4,5	bits 2,3	bits 0,1

The value in each field can be in the range 0-3, defining the assigned source or destination of each logical device. The values which can be assigned to each field are given in Table 5-3, below.

Table 5-3. IOBYTE Field Definitions

```
CONSOLE field (bits 0,1)
    0  - console is assigned to the console printer (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-86 utilities use the IOBYTE except for PIP which allows access to the physical devices, and STAT which allows logical-physical assignments to be made and displayed. In any case, you should omit the IOBYTE implementation until your basic CBIOS is fully implemented and tested, then add the IOBYTE to increase your facilities.

5.4 BIOS Subroutine Entry Points

The actions which must take place upon entry to each BIOS subroutine are given below. It should be noted that disk I/O is always performed through a sequence of calls on the various disk access subroutines. These setup the disk number to access, the track and sector on a particular disk, and the direct memory access (DMA) offset and segment addresses involved in the I/O operation. After all these parameters have been setup, 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 call to set the DMA segment base and a call to set the DMA offset 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.

The READ and WRITE subroutines should perform several retries (10 is standard) before reporting the error condition to the BDOS. 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.

Table 5-4. BIOS Subroutine Summary

Subroutine	Description
INIT	This subroutine is called directly by the CP/M-86 loader after the CPM.SYS file has been read into memory. The procedure is responsible for any hardware initialization not performed by the bootstrap loader, setting initial values for BIOS variables (including IOBYTE), printing a sign-on message, and initializing the interrupt vector to point to the BDOS offset (0B11H) and base. When this routine completes, it jumps to the CCP offset (0H). All segment registers should be initialized at this time to contain the base of the operating system.
WBOOT	This subroutine is called whenever a program terminates by performing a BDOS function #0 call. Some re-initialization of the hardware or software may occur here. When this routine completes, it jumps directly to the warm start entry point of the CCP (06H).
CONST	Sample the status of the currently assigned console device and return 0FFH in register AL if a character is ready to read, and 00H in register AL if no console characters are ready.

Table 5-4. (continued)

Subroutine	Description
CONIN	Read the next console character into register AL, 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 CL 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 have undesirable effects on the console device.
LIST	Send the character from register CL to the currently assigned listing device. The character is in ASCII with zero parity.
PUNCH	Send the character from register CL 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 AL 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 to the track 00 position. If your controller does not have a special feature for finding track 00, you can translate the call into a call to SETTRK with a parameter of 0.

Table 5-4. (continued)

Subroutine	Description
SELDSK	Select the disk drive given by register CL for further operations, where register CL contains 0 for drive A, 1 for drive B, and so on up to 15 for drive P (the standard CP/M-86 distribution version supports two drives). On each disk select, SELDSK must return in BX the base address of the selected drive's Disk Parameter Header. For standard floppy disk drives, the content of the header and associated tables does not change. The sample BIOS included with CP/M-86 called CBIOS contains an example program segment that performs the SELDSK function. If there is an attempt to select a non-existent drive, SELDSK returns BX=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 performed. This is due to the fact that disk select operations may take place without a subsequent disk operation and thus disk access may be substantially slower using some disk controllers. On entry to SELDSK it is possible to determine whether it is the first time the specified disk has been selected. Register DL, bit 0 (least significant bit) is a zero if the drive has not been previously selected. This information is of interest in systems which read configuration information from the disk in order to set up a dynamic disk definition table.
SETTRK	Register CX 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 CX 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 CX contains the translated sector number for subsequent disk accesses on the currently selected drive (see SECTRAN, below). 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.

Table 5-4. (continued)

Subroutine	Description
SETDMA	Register CX contains the DMA (disk memory access) offset for subsequent read or write operations. For example, if CX = 80H when SETDMA is called, then all subsequent read operations read their data into 80H through 0FFH offset from the current DMA segment base, and all subsequent write operations get their data from that address, until the next calls to SETDMA and SETDMAB occur. 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 offset and base 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 offset and segment base have been specified, the READ subroutine attempts to read one sector based upon these parameters, and returns the following error codes in register AL: 0 no errors occurred 1 non-recoverable error condition occurred Currently, CP/M-86 responds only to a zero or non-zero value as the return code. That is, if the value in register AL is 0 then CP/M-86 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 RETURN to ignore the error, or CONTROL-C to abort.
WRITE	Write the data from the currently selected DMA buffer 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 AL, with error recovery attempts as described above.
LISTST	Return the ready status of the list device. The value 00 is returned in AL if the list device is not ready to accept a character, and 0FFH if a character can be sent to the printer.

Table 5-4. (continued)

Subroutine	Description
SECTRAN	Performs logical to physical sector translation to improve the overall response of CP/M-86. Standard CP/M-86 systems are shipped with a "skew factor" of 6, where five physical sectors are skipped between sequential read or write operations. This skew factor allows enough time between sectors for most programs to load their buffers without missing the next sector. In computer systems that 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-86 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 CX. This logical sector number may range from 0 to the number of sectors -1. Sectran also receives a translate table offset in DX. The sector number is used as an index into the translate table, with the resulting physical sector number in BX. For standard systems, the tables and indexing code is provided in the CBIOS and need not be changed. If DX = 0000H no translation takes place, and CX is simply copied to BX before returning. Otherwise, SECTRAN computes and returns the translated sector number in BX. Note that SECTRAN is called when no translation is specified in the Disk Parameter Header.
SETDMAB	Register CX contains the segment base for subsequent DMA read or write operations. The BIOS will use the 128 byte buffer at the memory address determined by the DMA base and the DMA offset during read and write operations.
GETSEGB	Returns the address of the Memory Region Table (MRT) in BX. The returned value is the offset of the table relative to the start of the operating system. The table defines the location and extent of physical memory which is available for transient programs.

Table 5-4. (continued)

Subroutine	Description
	Memory areas reserved for interrupt vectors and the CP/M-86 operating system are not included in the MRT. The Memory Region Table takes the form: 8-bit MRT: R-Cnt 0: R-Base R-Length 1: R-Base R-Length . . . n: R-Base R-Length 16-bit 16-bit where R-Cnt is the number of Memory Region Descriptors (equal to n+1 in the diagram above), while R-Base and R-Length give the paragraph base and length of each physically contiguous area of memory. Again, the reserved interrupt locations, normally 0-3FFH, and the CP/M-86 operating system are not included in this map, because the map contains regions available to transient programs. If all memory is contiguous, the R-Cnt field is 1 and n = 0, with only a single Memory Region Descriptor which defines the region.
GETIOB	Returns the current value of the logical to physical input/output device byte (IOBYTE) in AL. This eight-bit value is used to associate physical devices with CP/M-86's four logical devices.
SETIOB	Use the value in CL to set the value of the IOBYTE stored in the BIOS.

The following section describes the exact layout and construction of the disk parameter tables referenced by various subroutines in the BIOS.

Section 6
BIOS Disk Definition Tables

Similar to CP/M-80, CP/M-86 is a table-driven operating system with a separate field-configurable Basic I/O System (BIOS). By altering specific subroutines in the BIOS presented in the previous section, CP/M-86 can be customized for operation on any RAM-based 8086 or 8088 microprocessor system.

The purpose of this section is to present the organization and construction of tables within the BIOS that define the characteristics of a particular disk system used with CP/M-86. These tables can be either hand-coded or automatically generated using the GENDEF utility provided with CP/M-86. The elements of these tables are presented below.

6.1 Disk Parameter Table Format

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 given in Table 6-1.

Table 6-1. Disk Parameter Header Elements

Element	Description
XLT	Offset 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).

Table 6-1. (continued)

Element	Description
DIRBUF	Offset of a 128 byte scratchpad area for directory operations within BDOS. All DPH's address the same scratchpad area.
DPB	Offset of a disk parameter block for this drive. Drives with identical disk characteristics address the same disk parameter block.
CSV	Offset of a scratchpad area used for software check for changed disks. This offset is different for each DPH.
ALV	Offset of a scratchpad area used by the BDOS to keep disk storage allocation information. This offset 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 offset of the DPH table relative to the beginning of the operating system.

A responsibility of the SELDSK subroutine, defined in the previous section, is to return the offset of the DPH from the beginning of the operating system for the selected drive. The following sequence of operations returns the table offset, with a 0000H returned if the selected drive does not exist.

```
NDISKS    EQU    4    ;NUMBER OF DISK DRIVES
......
SELDSK:
          ;SELECT DISK N GIVEN BY CL
          MOV    BX,0000H   ;READY FOR ERR
          CPM    CL,NDISKS  ;N BEYOND MAX DISKS?
          JNB    RETURN     ;RETURN IF SO
                            ;0 <= N < NDISKS
          MOV    CH,0       ;DOUBLE (N)
          MOV    BX,CX      ;BX = N
          MOV    CL,4       ;READY FOR * 16
          SHL    BX,CL      ;N = N * 16
          MOV    CX,OFFSET DPBASE
          ADD    BX,CX      ;DPBASE + N * 16
RETURN:   RET               ;BX = .DPH (N)
```

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. The fields are defined in Table 6-2.

Table 6-2. Disk Parameter Block Fields

Field	Definition
SPT	is the total number of sectors per track
BSH	is the data allocation block shift factor, determined by the data block allocation size.
BLM	is the block mask which is also 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

Table 6-2. (continued)

Field	Definition
AL0,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.

Although these table values are produced automatically by GENDEF, it is worthwhile reviewing the derivation of each field so that the values may be cross-checked when necessary. 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 you have selected a value for BLS, the values of BSH and BLM are shown in Table 6-3 below, where all values are in decimal.

Table 6-3. BSH and BLM Values for Selected BLS

BLS	BSH	BLM
1,024	3	7
2,048	4	15
4,096	5	31
8,192	6	63
16,384	7	127

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.

Table 6-4. Maximum EXM Values

BLS	DSM < 256	DSM > 255
1,024	0	N/A
2,048	1	0
4,096	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 one less than the total number of directory entries, which can take on a 16-bit value. The values of AL0 and AL1, however, are determined by DRM. The two values AL0 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 labeled AL0, and 15 corresponds to the low order bit of the byte labeled AL1. Each bit position reserves a data block for a number of directory entries, thus allowing a total of 16 data blocks to be assigned for directory entries (bits are assigned starting at 00 and filled to the right until position 15). Each directory entry occupies 32 bytes, as shown in Table 6-5.

Table 6-5. BLS and Number of Directory Entries

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 BIOS shown in Appendix D 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.

6.2 Table Generation Using GENDEF

The GENDEF utility supplied with CP/M-86 greatly simplifies the table construction process. GENDEF reads a file

```
x.DEF
```

containing the disk definition statements, and produces an output file

```
x.LIB
```

containing assembly language statements which define the tables necessary to support a particular drive configuration. The form of the GENDEF command is:

```
GENDEF x parameter list
```

where x has an assumed (and unspecified) filetype of DEF. The parameter list may contain zero or more of the symbols defined in Table 6-6.

Table 6-6. GENDEF Optional Parameters

Parameter	Effect
$C	Generate Disk Parameter Comments
$O	Generate DPBASE OFFSET $
$Z	Z80, 8080, 8085 Override
$COZ	(Any of the Above)

The C parameter causes GENDEF to produce an accompanying comment line, similar to the output from the "STAT DSK:" utility which describes the characteristics of each defined disk. Normally, the DPBASE is defined as

```
DPBASE  EQU  $
```

which requires a MOV CX,OFFSET DPBASE in the SELDSK subroutine shown above. For convenience, the $O parameter produces the definition

```
DPBASE  EQU  OFFSET $
```

allowing a MOV CX,DPBASE in SELDSK, in order to match your particular programming practices. The $Z parameter is included to override the standard 8086/8088 mode in order to generate tables acceptable for operation with Z80, 8080, and 8085 assemblers.

The disk definition contained within x.DEF is composed with the CP/M text editor, and consists of disk definition statements identical to those accepted by the DISKDEF macro supplied with CP/M-80 Version 2. A BIOS disk definition consists of the following sequence of statements:

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

Each statement is placed on a single line, with optional embedded comments between the keywords, numbers, and delimiters.

The DISKS statement defines the number of drives to be configured with your system, where n is an integer in the range 1 through 16. A series of DISKDEF statements 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 statements 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 at the end of your BIOS, before the start of uninitialized RAM.

The ENDEF (End of Diskdef) statement generates the necessary uninitialized RAM areas which are located beyond initialized RAM in your BIOS.

The form of the DISKDEF statement 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
dks     is the disk size in bls units
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 statement. 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 because there are fewer directory references. Also, logically connected data records are physically close on the disk. Further, each directory entry addresses more data and the amount of BIOS work 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 start or system reset has not occurred (when this situation is detected, CP/M-86 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 CP/M-80, version 1.4 which have been modified for higher density disks (typically double density). This parameter ensures that no directory compression takes place, which would cause incompatibilities with these non-standard CP/M 1.4 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 CP/M-80 Version 1.4, and upwardly compatible with CP/M-80 Version 2 implementations, is defined using the following statements:

```
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 a skew of 6 between sequential accesses, 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 statement generates n Disk Parameter Headers (DPH's), starting at the DPH table address DPBASE generated by the statement. Each disk header block contains sixteen bytes, as described above, and corresponds one-for-one to each of the defined drives. In the four drive standard system, for example, the DISKS statement 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 earlier in this section. The check and allocation vector addresses are generated by the ENDEF statement for inclusion 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 DX = 0000H, and simply returns the original logical sector from CX in the BX register. 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 statement call:

```
XLT0    EQU    OFFSET $
        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 statement, 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 operating system memory. The size of the uninitialized RAM area is determined by EQU statements generated by the ENDEF statement. For a standard four-drive system, the ENDEF statement might produce

```
1C72 =         BEGDAT EQU OFFSET $
               (data areas)
1DB0 =         ENDDAT EQU OFFSET $
013C =         DATSIZ EQU OFFSET $-BEGDAT
```

which indicates that uninitialized RAM begins at offset 1C72H, ends at 1DB0H-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 comment included in the LIB file by the $C parameter to GENCMD will match the output from STAT. 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
```

6.3 GENDEF Output

GENDEF produces a listing of the statements included in the DEF file at the user console (CONTROL-P can be used to obtain a printed listing, if desired). Each source line is numbered, and any errors are shown below the line in error, with a "?" beneath the item which caused the condition. The source errors produced by GENCMD are listed in Table 6-7, followed by errors that can occur when producing input and output files in Table 6-8.

Table 6-7. GENDEF Source Error Messages

Message	Meaning
Bad Val	More than 16 disks defined in DISKS statement.
Convert	Number cannot be converted, must be constant in binary, octal, decimal, or hexadecimal as in ASM-86.
Delimit	Missing delimiter between parameters.
Duplic	Duplicate definition for a disk drive.
Extra	Extra parameters occur at the end of line.
Length	Keyword or data item is too long.
Missing	Parameter required in this position.
No Disk	Referenced disk not previously defined.
No Stmt	Statement keyword not recognized.
Numeric	Number required in this position
Range	Number in this position is out of range.
Too Few	Not enough parameters provided.
Quote	Missing end quote on current line.

Table 6-8. GENDEF Input and Output Error Messages

Message	Meaning
Cannot Close ".LIB" File	LIB file close operation unsuccessful, usually due to hardware write protect.
"LIB" Disk Full	No space for LIB file.
No Input File Present	Specified DEF file not found.
No ".LIB" Directory Space	Cannot create LIB file due to too many files on LIB disk.
Premature End-of-File	End of DEF file encountered unexpectedly.

Given the file TWO.DEF containing the following statements

```
disks   2
diskdef 0,1,26,6,2048,256,128,128,2
diskdef 1,1,58,,2048,1024,300,0,2
endef
```

the command

```
gencmd two $c
```

produces the console output

```
DISKDEF Table Generator, Vers 1.0
1             DISKS   2
2             DISKDEF 0,1,58,,2048,256,128,128,2
3             DISKDEF 1,1,58,,2048,1024,300,0,2
4             ENDEF
No Error(s)
```

The resulting TWO.LIB file is brought into the following skeletal assembly language program, using the ASM-86 INCLUDE directive. The ASM-86 output listing is truncated on the right, but can be easily reproduced using GENDEF and ASM-86.

```
                                ;       Sample Program Including TWO.LI
                                ;
                                SELDSK:
                                ;       ....
 0000 B9 03 00                          MOV     CX,OFFSET DPBASE
                                ;       ....
=                                       INCLUDE TWO.LIB
=                               ;               DISKS   2
=   0003                        dpbase  equ     $               ;Base o
= 0003 32 00 00 00              dpe0    dw      xlt0,0000h      ;Transl
= 0007 00 00 00 00                      dw      0000h,0000h     ;Scratc
= 000B 5B 00 23 00                      dw      dirbuf,dpb0     ;Dir Bu
= 000F FB 00 DB 00                      dw      csv0,alv0       ;Check,
= 0013 00 00 00 00              dpe1    dw      xlt1,0000h      ;Transl
= 0017 00 00 00 00                      dw      0000h,0000h     ;Scratc
= 001B 5B 00 4C 00                      dw      dirbuf,dpb1     ;Dir Bu
= 001F 9B 01 1B 01                      dw      csv1,alv1       ;Check,
=                               ;               DISKDEF 0,1,26,6,2048,2
=                               ;
=                               ;       Disk 0 is CP/M 1.4 Single Densi
=                               ;        4096:  128 Byte Record Capacit
=                               ;         512:  Kilobyte Drive  Capacit
=                               ;         128:  32 Byte Directory Entri
=                               ;         128:  Checked Directory Entri
=                               ;         256:  Records / Extent
=                               ;          16:  Records / Block
=                               ;          26:  Sectors / Track
=                               ;           2:  Reserved  Tracks
=                               ;           6:  Sector Skew Factor
=                               ;
=   0023                        dpb0    equ     offset $        ;Disk P
= 0023 1A 00                            dw      26              ;Sector
= 0025 04                               db      4               ;Block
= 0026 0F                               db      15              ;Block
= 0027 01                               db      1               ;Extnt
= 0028 FF 00                            dw      255             ;Disk S
= 002A 7F 00                            dw      127             ;Direct
= 002C C0                               db      192             ;Alloc0
= 002D 00                               db      0               ;Alloc1
= 002E 20 00                            dw      32              ;Check
= 0030 02 00                            dw      2               ;Offset
=   0032                        xlt0    equ     offset $        ;Transl
= 0032 01 07 0D 13                      db      1,7,13,19
= 0036 19 05 0B 11                      db      25,5,11,17
= 003A 17 03 09 0F                      db      23,3,9,15
= 003E 15 02 08 0E                      db      21,2,8,14
= 0042 14 1A 06 0C                      db      20,26,6,12
= 0046 12 18 04 0A                      db      18,24,4,10
= 004A 10 16                            db      16,22
=   0020                        als0    equ     32              ;Alloca
=   0020                        css0    equ     32              ;Check
=                               ;               DISKDEF 1,1,58,,2048,10
=                               ;
=                               ;       Disk 1 is CP/M 1.4 Single Densi
=                               ;       16384:  128 Byte Record Capacit
```

```
=                          ;       2048:  Kilobyte Drive  Capacit
=                          ;        300:  32 Byte Directory Entri
=                          ;          0:  Checked Directory Entri
=                          ;        128:  Records / Extent
=                          ;         16:  Records / Block
=                          ;         58:  Sectors / Track
=                          ;          2:  Reserved  Tracks
=                          ;
=    004C                  dpbl    equ    offset $        ;Disk P
= 004C 3A 00                       dw     58              ;Sector
= 004E 04                          db     4               ;Block
= 004F 0F                          db     15              ;Block
= 0050 00                          db     0               ;Extnt
= 0051 FF 03                       dw     1023            ;Disk S
= 0053 2B 01                       dw     299             ;Direct
= 0055 F8                          db     248             ;Alloc0
= 0056 00                          db     0               ;Alloc1
= 0057 00 00                       dw     0               ;Check
= 0059 02 00                       dw     2               ;Offset
=    0000                  xltl    equ    0               ;No Tra
=    0080                  alsl    equ    128             ;Alloca
=    0000                  cssl    equ    0               ;Check
=                          ;              ENDEF
=                          ;
=                          ;       Uninitialized Scratch Memory Fo
=                          ;
=    005B                  begdat  equ    offset $        ;Start
= 005B                     dirbuf  rs     128             ;Direct
= 00DB                     alv0    rs     als0            ;Alloc
= 00FB                     csv0    rs     css0            ;Check
= 011B                     alvl    rs     alsl            ;Alloc
= 019B                     csvl    rs     cssl            ;Check
=    019B                  enddat  equ    offset $        ;End of
=    0140                  datsiz  equ    offset $-begdat ;Size o
= 019B 00                          db     0               ;Marks
                                   END
```

Section 7
CP/M-86 Bootstrap and Adaption Procedures

This section describes the components of the standard CP/M-86 distribution disk, the operation of each component, and the procedures to follow in adapting CP/M-86 to non-standard hardware.

CP/M-86 is distributed on a single-density IBM compatible 8" diskette using a file format which is compatible with all previous CP/M-80 operating systems. In particular, the first two tracks are reserved for operating system and bootstrap programs, while the remainder of the diskette contains directory information which leads to program and data files. CP/M-86 is distributed for operation with the Intel SBC 86/12 single-board computer connected to floppy disks through an Intel 204 Controller. The operation of CP/M-86 on this configuration serves as a model for other 8086 and 8088 environments, and is presented below.

The principal components of the distribution system are listed below:

- The 86/12 Bootstrap ROM (BOOT ROM)
- The Cold Start Loader (LOADER)
- The CP/M-86 System (CPM.SYS)

When installed in the SBC 86/12, the BOOT ROM becomes a part of the memory address space, beginning at byte location 0FF000H, and receives control when the system reset button is depressed. In a non-standard environment, the BOOT ROM is replaced by an equivalent initial loader and, therefore, the ROM itself is not included with CP/M-86. The BOOT ROM can be obtained from Digital Research or, alternatively, it can be programmed from the listing given in Appendix C or directly from the source file which is included on the distribution disk as BOOT.A86. The responsibility of the BOOT ROM is to read the LOADER from the first two system tracks into memory and pass program control to the LOADER for execution.

7.1 The Cold Start Load Operation

The LOADER program is a simple version of CP/M-86 that contains sufficient file processing capability to read CPM.SYS from the system disk to memory. When LOADER completes its operation, the CPM.SYS program receives control and proceeds to process operator input commands.

Both the LOADER and CPM.SYS programs are preceded by the standard CMD header record. The 128-byte LOADER header record contains the following single group descriptor.

G-Form	G-Length	A-Base	G-Min	G-Max
1	xxxxxxxxx	0400	xxxxxxx	xxxxxxx
8b	16b	16b	16b	16b

where G-Form = 1 denotes a code group, "x" fields are ignored, and A-Base defines the paragraph address where the BOOT ROM begins filling memory (A-Base is the word value which is offset three bytes from the beginning of the header). Note that since only a code group is present, an 8080 memory model is assumed. Further, although the A-Base defines the base paragraph address for LOADER (byte address 04000H), the LOADER can, in fact be loaded and executed at any paragraph boundary that does not overlap CP/M-86 or the BOOT ROM.

The LOADER itself consists of three parts: the Load CPM program (LDCPM), the Loader Basic Disk System (LDBDOS), and the Loader Basic I/O System (LDBIOS). Although the LOADER is setup to initialize CP/M-86 using the Intel 86/12 configuration, the LDBIOS can be field-altered to account for non-standard hardware using the same entry points described in a previous section for BIOS modification. The organization of LOADER is shown in Figure 7-1 below:

```
                      +--------------------------+
                      | GD#1 0 ///////////////// |
                      +--------------------------+
CS DS ES SS 0000H:    | JMP 1200H |              |
                      +-----------+              |
                      |         (LDCPM)          |
                      |             +------------+
                      |             | JMPF CPM   |
                      +--------------------------+
             0400H:   |         (LDBDOS)         |
                      +--------------------------+
             1200H:   |  JMP INIT                |
                      |  ... ....                |
                      |  JMP SETIOB              |
                      |                          |
                      |  INIT: .. JMP 0003H      |
                      |         (LDBIOS)         |
                      +--------------------------+
             1700H:
```

Figure 7-1. LOADER Organization

Byte offsets from the base registers are shown at the left of the diagram. GD#1 is the Group Descriptor for the LOADER code group described above, followed immediately by a "0" group terminator. The entire LOADER program is read by the BOOT ROM, excluding the header record, starting at byte location 04000H as given by the A-Field. Upon completion of the read, the BOOT ROM passes control to location 04000H where the LOADER program commences execution. The JMP 1200H instruction at the base of LDCPM transfers control to the beginning of the LDBIOS where control then transfers to the INIT subroutine. The subroutine starting at INIT performs device initialization, prints a sign-on message, and transfers back to the LDCPM program at byte offset 0003H. The LDCPM module opens the CPM.SYS file, loads the CP/M-86 system into memory and transfers control to CP/M-86 through the JMPF CPM instruction at the end of LDCPM execution, thus completing the cold start sequence.

The files LDCPM.H86 and LDBDOS.H86 are included with CP/M-86 so that you can append your own modified LDBIOS in the construction of a customized loader. In fact, BIOS.A86 contains a conditional assembly switch, called "loader_bios," which, when enabled, produces the distributed LDBIOS. The INIT subroutine portion of LDBIOS is listed in Appendix C for reference purposes. To construct a custom LDBIOS, modify your standard BIOS to start the code at offset 1200H, and change your initialization subroutine beginning at INIT to perform disk and device initialization. Include a JMP to offset 0003H at the end of your INIT subroutine. Use ASM-86 to assemble your LDBIOS.A86 program:

```
ASM86 LDBIOS
```

to produce the LDBIOS.H86 machine code file. Concatenate the three LOADER modules using PIP:

```
PIP LOADER.H86=LDCPM.H86,LDBDOS.H86,LDBIOS.H86
```

to produce the machine code file for the LOADER program. Although the standard LOADER program ends at offset 1700H, your modified LDBIOS may differ from this last address with the restriction that the LOADER must fit within the first two tracks and not overlap CP/M-86 areas. Generate the command (CMD) file for LOADER using the GENCMD utility:

```
GENCMD LOADER 8080 CODE[A400]
```

resulting in the file LOADER.CMD with a header record defining the 8080 Memory Model with an absolute paragraph address of 400H, or byte address 4000H. Use DDT to read LOADER.CMD to location 900H in your 8080 system. Then use the 8080 utility SYSGEN to copy the loader to the first two tracks of a disk.

```
A>DDT
-ILOADER.CMD
-R800
-^C
A>SYSGEN
SOURCE DRIVE NAME (or return to skip) <cr>
DESTINATION DRIVE NAME (or return to skip) B
```

Alternatively, if you have access to an operational CP/M-86 system, the command

```
LDCOPY LOADER
```

copies LOADER to the system tracks. You now have a diskette with a LOADER program which incorporates your custom LDBIOS capable of reading the CPM.SYS file into memory. For standardization, we assume LOADER executes at location 4000H. LOADER is statically relocatable, however, and its operating address is determined only by the value of A-Base in the header record.

You must, of course, perform the same function as the BOOT ROM to get LOADER into memory. The boot operation is usually accomplished in one of two ways. First, you can program your own ROM (or PROM) to perform a function similar to the BOOT ROM when your computer's reset button is pushed. As an alternative, most controllers provide a power-on "boot" operation that reads the first disk sector into memory. This one-sector program, in turn, reads the LOADER from the remaining sectors and transfers to LOADER upon completion, thereby performing the same actions as the BOOT ROM. Either of these alternatives is hardware-specific, so you'll need to be familiar with the operating environment.

7.2 Organization of CPM.SYS

The CPM.SYS file, read by the LOADER program, consists of the CCP, BDOS, and BIOS in CMD file format, with a 128-byte header record similar to the LOADER program:

G-Form	G-Length	A-Base	G-Min	G-Max
1	xxxxxxxxx	040	xxxxxxx	xxxxxxx
8b	16b	16b	16b	16b

where, instead, the A-Base load address is paragraph 040H, or byte address 0400H, immediately following the 8086 interrupt locations. The entire CPM.SYS file appears on disk as shown in Figure 7-2.

```
                               +----+-+-------------+
                               |GD#1|0|/////////////|
                               +----+-+-------------+
(0040:0) CS DS ES SS 0000H:    |                    |
                               |   (CCP and BDOS)   |
                               |                    |
               (0040:) 2500H:  | JMP INIT           |
                               | ... ....           |
                               | JMP SETIOB         |
                               |       (BIOS)       |
                               | INIT: .. JMP 0000H |
                               +--------------------+
               (0040:) 2A00H:
```

Figure 7-2. CPM.SYS File Organization

where GD#1 is the Group Descriptor containing the A-Base value followed by a "0" terminator. The distributed 86/12 BIOS is listed in Appendix D, with an "include" statement that reads the SINGLES.LIB file containing the disk definition tables. The SINGLES.LIB file is created by GENDEF using the SINGLES.DEF statements shown below:

```
disks 2
diskdef 0,1,26,6,1024,243,64,64,2
diskdef 1,0
endef
```

The CPM.SYS file is read by the LOADER program beginning at the address given by A-Base (byte address 0400H), and control is passed to the INIT entry point at offset address 2500H. Any additional initialization, not performed by LOADER, takes place in the INIT subroutine and, upon completion, INIT executes a JMP 0000H to begin execution of the CCP. The actual load address of CPM.SYS is determined entirely by the address given in the A-Base field which can be changed if you wish to execute CP/M-86 in another region of memory. Note that the region occupied by the operating system must be excluded from the BIOS memory region table.

Similar to the LOADER program, you can modify the BIOS by altering either the BIOS.A86 or skeletal CBIOS.A86 assembly language files which are included on your source disk. In either case, create a customized BIOS which includes your specialized I/O drivers, and assemble using ASM-86:

```
ASM86 BIOS
```

to produce the file BIOS.H86 containing your BIOS machine code.

Concatenate this new BIOS to the CPM.H86 file on your distribution disk:

```
PIP CPMX.H86 = CPM.H86,BIOS.H86
```

The resulting CPMX hex file is then converted to CMD file format by executing

```
GENCMD CPMX 8080 CODE[A40]
```

in order to produce the CMD memory image with A-Base = 40H. Finally, rename the CPMX file using the command

```
REN CPM.SYS = CPMX.CMD
```

and place this file on your 8086 system disk. Now the tailoring process is complete: you have replaced the BOOT ROM by either your own customized BOOT ROM, or a one-sector cold start loader which brings the LOADER program, with your custom LDBIOS, into memory at byte location 04000H. The LOADER program, in turn, reads the CPM.SYS file, with your custom BIOS, into memory at byte location 0400H. Control transfers to CP/M-86, and you are up and operating. CP/M-86 remains in memory until the next cold start operation takes place.

You can avoid the two-step boot operation if you construct a non-standard disk with sufficient space to hold the entire CPM.SYS file on the system tracks. In this case, the cold start brings the CP/M-86 memory image into memory at the location given by A-Base, and control transfers to the INIT entry point at offset 2500H. Thus, the intermediate LOADER program is eliminated entirely, although the initialization found in the LDBIOS must, of course, take place instead within the BIOS.

Since ASM-86, GENCMD and GENDEF are provided in both COM and CMD formats, either CP/M-80 or CP/M-86 can be used to aid the customizing process. If CP/M-80 or CP/M-86 is not available, but you have minimal editing and debugging tools, you can write specialized disk I/O routines to read and write the system tracks, as well as the CPM.SYS file.

The two system tracks are simple to access, but the CPM.SYS file is somewhat more difficult to read. CPM.SYS is the first file on the disk and thus it appears immediately following the directory on the diskette. The directory begins on the third track, and occupies the first sixteen logical sectors of the diskette, while the CPM.SYS is found starting at the seventeenth sector. Sectors are "skewed" by a factor of six beginning with the directory track (the system tracks are sequential), so that you must load every sixth sector in reading the CPM.SYS file. Clearly, it is worth the time and effort to use an existing CP/M system to aid the conversion process.

Appendix A
Sector Blocking and Deblocking

Upon each call to the BIOS WRITE entry point, the CP/M-86 BDOS includes information that 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. This appendix presents a general-purpose algorithm that can be included within your BIOS and that uses the BDOS information to perform the operations automatically.

Upon each call to WRITE, the BDOS provides the following information in register CL:

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.

This appendix lists the blocking and deblocking algorithm in skeletal form (the file is included on your CP/M-86 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 24 of Appendix F define the mapping between CP/M and the host system, and must be changed if other than the sample host system is involved.

The SELDSK entry point clears the host buffer flag whenever a new disk is logged-in. Note that although the SELDSK entry point computes and returns the Disk Parameter Header address, it does not physically select the host disk at this point (it is selected later at READHST or WRITEHST). Further, SETTRK, SETSEC, and SETDMA simply store the values, but do not take any other action at this point. SECTRAN performs a trivial function of returning the physical sector number.

The principal entry points are READ and WRITE. 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.

```
;*****************************************************
;*                                                   *
;*          Sector Blocking / Deblocking             *
;*                                                   *
;* This algorithm is a direct translation of the     *
;* CP/M-80 Version, and is included here for refer-  *
;* ence purposes only.  The file DEBLOCK.LIB is in-  *
;* cluded on your CP/M-86 disk, and should be used   *
;* for actual applications.  You may wish to contact *
;* Digital Research for notices of updates.          *
;*                                                   *
;*****************************************************
;
;*****************************************************
;*                                                   *
;*          CP/M to host disk constants              *
;*                                                   *
;* (This example is setup for CP/M block size of 16K *
;* with a host sector size of 512 bytes, and 12 sec- *
;* tors per track.  Blksiz, hstsiz, hstspt, hstblk   *
;* and secshf may change for different hardware.)    *
;*****************************************************
una     equ     byte ptr [BX]   ;name for byte at BX
;
blksiz  equ     16384           ;CP/M allocation size
hstsiz  equ     512             ;host disk sector size
hstspt  equ     12              ;host disk sectors/trk
hstblk  equ     hstsiz/128      ;CP/M sects/host buff
;
;*****************************************************
;*                                                   *
;* secshf is log2(hstblk), and is listed below for   *
;* values of hstsiz up to 2048.                      *
;*                                                   *
;*          hstsiz    hstblk    secshf               *
;*             256         2         1               *
;*             512         4         2               *
;*            1024         8         3               *
;*            2048        16         4               *
;*                                                   *
```

```
;*****************************************************
secshf  equ     2               ;log2(hstblk)
cpmspt  equu    hstblk * hstspt ;CP/M sectors/track
secmsk  equ     hstblk-1        ;sector mask
;
;*****************************************************
;*                                                   *
;*         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 BIOS entry points given below show the   *
;*      code which is relevant to deblocking only.   *
;*                                                   *
;*****************************************************
seldsk:
        ;select disk
        ;is this the first activation of the drive?
        test DL,1               ;lsb = 0?
        jnz selset
        ;this is the first activation, clear host buff
        mov hstact,0
        mov unacnt,0
selset:
        mov al,cl ! cbw         ;put in AX
        mov sekdsk,al           ;seek disk number
        mov cl,4 ! shl al,cl    ;times 16
        add ax,offset dpbase
        mov bx,ax
        ret
;
home:
        ;home the selected disk
        mov al,hstwrt           ;check for pending write
        test al,al
        jnz homed
        mov hstact,0            ;clear host active flag
homed:
        mov cx,0                ;now, set track zero
;       (continue HOME routine)
        ret
;
settrk:
        ;set track given by registers CX
        mov sektrk,CX           ;track to seek
        ret
;
setsec:
        ;set sector given by register cl
        mov seksec,cl           ;sector to seek
```

```
        ret
;
setdma:
        ;set dma address given by CX
        mov dma_off,CX
        ret
;
setdmab:
        ;set segment address given by CX
        mov dma_seg,CX
        ret
;
sectran:
        ;translate sector number CX with table at [DX]
        test DX,DX      ;test for hard skewed
        jz notran       ;(blocked must be hard skewed)
        mov BX,CX
        add BX,DX
        mov BL,[BX]
        ret
no_tran:
        ;hard skewed disk, physical = logical sector
        mov BX,CX
        ret
;
read:
        ;read the selected CP/M sector
        mov unacnt,0            ;clear unallocated counter
        mov readop,1            ;read operation
        mov rsflag,1            ;must read data
        mov wrtype,wrual        ;treat as unalloc
        jmp rwoper              ;to perform the read
;
write:
        ;write the selected CP/M sector
        mov readop,0            ;write operation
        mov wrtype,cl
        cmp cl,wrual            ;write unallocated?
        jnz chkuna              ;check for unalloc
;
;       write to unallocated, set parameters
;
        mov unacnt,(blksiz/128) ;next unalloc recs
        mov al,sekdsk           ;disk to seek
        mov unadsk,al           ;unadsk = sekdsk
        mov ax,sektrk
        mov unatrk,ax           ;unatrk = sektrk
        mov al,seksec
        mov unasec,al           ;unasec = seksec
;
chkuna:
        ;check for write to unallocated sector
;
        mov bx,offset unacnt    ;point "UNA" at UNACNT
        mov al,una ! test al,al ;any unalloc remain?
```

```
                jz alloc                ;skip if not
        ;
        ;       more unallocated records remain
                dec al                  ;unacnt = unacnt-1
                mov una,al
                mov al,sekdsk           ;same disk?
                mov BX,offset unadsk
                cmp al,una              ;sekdsk = unadsk?
                jnz alloc               ;skip if not
        ;
        ;       disks are the same
                mov AX, unatrk
                cmp AX, sektrk
                jnz alloc               ;skip if not
        ;
        ;       tracks are the same
                mov al,seksec           ;same sector?
        ;
                mov BX,offset unasec    ;point una at unasec
        ;
                cmp al,una              ;seksec = unasec?
                jnz alloc               ;skip if not
        ;
        ;       match, move to next sector for future ref
                inc una                 ;unasec = unasec+1
                mov al,una              ;end of track?
                cmp al,cpmspt           ;count CP/M sectors
                jb noovf                ;skip if below
        ;
        ;       overflow to next track
                mov una,0               ;unasec = 0
                inc unatrk              ;unatrk=unatrk+1
        ;
        noovf:
                ;match found, mark as unnecessary read
                mov rsflag,0            ;rsflag = 0
                jmps rwoper             ;to perform the write
        ;
        alloc:
                ;not an unallocated record, requires pre-read
                mov unacnt,0            ;unacnt = 0
                mov rsflag,1            ;rsflag = 1
                                        ;drop through to rwoper
        ;
        ;*****************************************************
        ;*                                                   *
        ;*      Common code for READ and WRITE follows       *
        ;*                                                   *
        ;*****************************************************
        rwoper:
                ;enter here to perform the read/write
                mov erflag,0            ;no errors (yet)
                mov al, seksec          ;compute host sector
                mov cl, secshf
                shr al,cl
```

```
        mov sekhst,al           ;host sector to seek
;
;       active host sector?
        mov al,1
        xchg al,hstact          ;always becomes 1
        test al,al              ;was it already?
        jz filhst               ;fill host if not
;
;       host buffer active, same as seek buffer?
        mov al,sekdsk
        cmp al,hstdsk           ;sekdsk = hstdsk?
        jnz nomatch
;
;       same disk, same track?
        mov ax,hsttrk
        cmp ax,sektrk           ;host track same as seek track
        jnz nomatch
;
;       same disk, same track, same buffer?
        mov al,sekhst
        cmp al,hstsec           ;sekhst = hstsec?
        jz match                ;skip if match
nomatch:
        ;proper disk, but not correct sector
        mov al, hstwrt
        test al,al              ;"dirty" buffer ?
        jz filhst               ;no, don't need to write
        call writehst           ;yes, clear host buff
;       (check errors here)
;
filhst:
        ;may have to fill the host buffer
        mov al,sekdsk ! mov hstdsk,al
        mov ax,sektrk ! mov hsttrk,ax
        mov al,sekhst ! mov hstsec,al
        mov al,rsflag
        test al,al              ;need to read?
        jz filhstl
;
        call readhst            ;yes, if 1
;       (check errors here)
;
filhstl:
        mov hstwrt,0            ;no pending write
;
match:
        ;copy data to or from buffer depending on "readop"
        mov al,seksec           ;mask buffer number
        and ax,secmsk           ;least signif bits are masked
        mov cl, 7 ! shl ax,cl   ;shift left 7 (* 128 = 2**7)
;
;       ax has relative host buffer offset
;
        add ax,offset hstbuf    ;ax has buffer address
        mov si,ax               ;put in source index register
```

```
        mov di,dma_off          ;user buffer is dest if readop
;
        push DS ! push ES       ;save segment registers
;
        mov ES,dma_seg          ;set destseg to the users seg
                                ;SI/DI and DS/ES is swapped
                                ;if write op
        mov cx,128/2            ;length of move in words
        mov al,readop
        test al,al              ;which way?
        jnz     rwmove          ;skip if read
;
;       write operation, mark and switch direction
        mov hstwrt,1            ;hstwrt = 1 (dirty buffer now)
        xchg si,di              ;source/dest index swap
        mov ax,DS
        mov ES,ax
        mov DS,dma_seg          ;setup DS,ES for write
;
rwmove:
        cld ! rep movs AX,AX    ;move as 16 bit words
        pop ES ! pop DS         ;restore segment registers
;
;       data has been moved to/from host buffer
        cmp wrtype,wrdir        ;write type to directory?
        mov al,erflag           ;in case of errors
        jnz return_rw           ;no further processing
;
;       clear host buffer for directory write
        test al,al              ;errors?
        jnz return_rw           ;skip if so
        mov hstwrt,0            ;buffer written
        call writehst
        mov al,erflag
return_rw:
        ret
;
;*****************************************************
;*                                                   *
;* WRITEHST performs the physical write to the host  *
;* disk, while READHST reads the physical disk.      *
;*                                                   *
;*****************************************************
writehst:
        ret
;
readhst:
        ret
;
;*****************************************************
;*                                                   *
;* Use the GENDEF utility to create disk def tables  *
;*                                                   *
;*****************************************************
dpbase  equu    offset $
```

```
;       disk parameter tables go here
;
;*****************************************************
;*                                                   *
;* Uninitialized RAM areas follow, including the     *
;* areas created by the GENDEF utility listed above. *
;*                                                   *
;*****************************************************
sek_dsk rb      1               ;seek disk number
sek_trk rw      1               ;seek track number
sek_sec rb      1               ;seek sector number
;
hst_dsk rb      1               ;host disk number
hst_trk rw      1               ;host track number
hst_sec rb      1               ;host sector number
;
sek_hst rb      1               ;seek shr secshf
hst_act rb      1               ;host active flag
hst_wrt rb      1               ;host written flag
;
una_cnt rb      1               ;unalloc rec cnt
una_dsk rb      1               ;last unalloc disk
una_trk rw      1               ;last unalloc track
una_sec rb      1               ;last unalloc sector
;
erflag  rb      1               ;error reporting
rsflag  rb      1               ;read sector flag
readop  rb      1               ;1 if read operation
wrtype  rb      1               ;write operation type
dma_seg rw      1               ;last dma segment
dma_off rw      1               ;last dma offset
hstbuf  rb      hstsiz          ;host buffer
        end
```

Appendix B
Sample Random Access Program

This appendix contains a rather extensive and complete example of random access operation. The program listed here performs the simple function of reading or writing random records upon command from the terminal. Given that the program has been created, assembled, and placed into a file labelled RANDOM.CMD, the CCP level command:

    RANDOM X.DAT

starts the test program. The program looks for a file by the name X.DAT (in this particular case) and, if found, proceeds to prompt the console for input. If not found, the file is created before the prompt is given. Each prompt takes the form

    next command?

and is followed by operator input, terminated by a carriage return. The input commands take the form

    nW   nR   Q

where n is an integer value in the range 0 to 65535, and W, R, and Q are simple command characters corresponding to random write, random read, and quit processing, respectively. If the W command is issued, the RANDOM program issues the prompt

    type data:

The operator then responds by typing up to 127 characters, followed by a carriage return. RANDOM then writes the character string into the X.DAT file at record n. If the R command is issued, RANDOM reads record number n and displays the string value at the console. If the Q command is issued, the X.DAT file is closed, and the program returns to the console command processor. The only error message is

    error, try again

The program begins with an initialization section where the input file is opened or created, followed by a continuous loop at the label "ready" where the individual commands are interpreted. The default file control block at offset 005CH and the default buffer at offset 0080H are used in all disk operations. The utility subroutines then follow, which contain the principal input line processor, called "readc." This particular program shows the elements of random access processing, and can be used as the basis for further program development. In fact, with some work, this program could evolve into a simple data base management system.

One could, for example, assume a standard record size of 128 bytes, consisting of arbitrary fields within the record. A program called GETKEY, could be developed which first reads a sequential file and extracts a specific field defined by the operator. For example, the command

    GETKEY NAMES.DAT LASTNAME 10 20

would cause GETKEY to read the data base file NAMES.DAT and extract the "LASTNAME" field from each record, starting at position 10 and ending at character 20. GETKEY builds a table in memory consisting of each particular LASTNAME field, along with its 16-bit record number location within the file. The GETKEY program then sorts this list, and writes a new file, called LASTNAME.KEY, which is an alphabetical list of LASTNAME fields with their corresponding record numbers. (This list is called an "inverted index" in information retrieval parlance.)

Rename the program shown above as QUERY, and enhance it a bit so that it reads a sorted key file into memory. The command line might appear as:

    QUERY NAMES.DAT LASTNAME.KEY

Instead of reading a number, the QUERY program reads an alphanumeric string which is a particular key to find in the NAMES.DAT data base. Since the LASTNAME.KEY list is sorted, you can find a particular entry quite rapidly by performing a "binary search," similar to looking up a name in the telephone book. That is, starting at both ends of the list, you examine the entry halfway in between and, if not matched, split either the upper half or the lower half for the next search. You'll quickly reach the item you're looking for (in log2(n) steps) where you'll find the corresponding record number. Fetch and display this record at the console, just as we have done in the program shown above.

At this point you're just getting started. With a little more work, you can allow a fixed grouping size which differs from the 128 byte record shown above. This is accomplished by keeping track of the record number as well as the byte offset within the record. Knowing the group size, you randomly access the record containing the proper group, offset to the beginning of the group within the record read sequentially until the group size has been exhausted.

Finally, you can improve QUERY considerably by allowing boolean expressions which compute the set of records which satisfy several relationships, such as a LASTNAME between HARDY and LAUREL, and an AGE less than 45. Display all the records which fit this description. Finally, if your lists are getting too big to fit into memory, randomly access your key files from the disk as well.

```
;
;*****************************************************
;*                                                   *
;*     Sample Random Access Program for CP/M-86      *
;*                                                   *
;*****************************************************
;
;     BDOS Functions
;
coninp  equ     1       ;console input function
conout  equ     2       ;console output function
pstring equ     9       ;print string until '$'
rstring equ     10      ;read console buffer
version equ     12      ;return version number
openf   equ     15      ;file open function
closef  equ     16      ;close function
makef   equ     22      ;make file function
readr   equ     33      ;read random
writer  equ     34      ;write random
;
;    Equates for non graphic characters
cr      equ     0dh     ;carriage return
lf      equ     0ah     ;line feed
;
;
;   load SP, ready file for random access
;
        cseg
        pushf                   ;push flags in CCP stack
        pop     ax              ;save flags in AX
        cli                     ;disable interrupts
        mov     bx,ds           ;set SS register to base
        mov     ss,bx           ;set SS, SP with interru
        mov     sp,offset stack ;    for 80888
        push    ax              ;restore the flags
        popf
;
;       CP/M-86 initial release returns the file
;       system version number of 2.2: check is
;       shown below for illustration purposes.
;
        mov     cl,version
        call    bdos
        cmp     al,20h          ;version 2.0 or later?
        jnb     versok
;       bad version, message and go back
        mov     dx,offset badver
        call    print
        jmp     abort
;
versok:
;       correct version for random access
        mov     cl,openf        ;open default fct
        mov     dx,offset fcb
        call    bdos
```

```
         inc     al              ;err 255 becomes zero
         jnz     ready
;
;        cannot open file, so create it
         mov     cl,makef
         mov     dx,offset fcb
         call    bdos
         inc     al              ;err 255 becomes zero
         jnz     ready
;
;        cannot create file, directory full
         mov     dx,offset nospace
         call    print
         jmp     abort           ;back to ccp
;
;   loop back to "ready" after each command
;
ready:
;        file is ready for processing
;
         call    readcom         ;read next command
         mov     ranrec,dx       ;store input record#
         mov     ranovf,0h       ;clear high byte if set
         cmp     al,'Q'          ;quit?
         jnz     notq
;
;        quit processing, close file
         mov     cl,closef
         mov     dx,offset fcb
         call    bdos
         inc     al      ;err 255 becomes 0
         jz      error   ;error message, retry
         jmps    abort   ;back to ccp
;
;
;  end of quit command, process write
;
;
notq:
;        not the quit command, random write?
         cmp     al,'W'
         jnz     notw
;
;        this is a random write, fill buffer until cr
         mov     dx,offset datmsg
         call    print           ;data prompt
         mov     cx,127          ;up to 127 characters
         mov     bx,offset buff  ;destination
rloop:   ;read next character to buff
         push    cx              ;save loop conntrol
         push    bx              ;next destination
         call    getchr          ;character to AL
         pop     bx              ;restore destination
         pop     cx              ;restore counter
         cmp     al,cr           ;end of line?
```

```
                jz      erloop
;               not end, store character
                mov     byte ptr [bx],al
                inc     bx                      ;next to fill
                loop    rloop                   ;decrement cx ..loop if
erloop:
;               end of read loop, store 00
                mov     byte ptr [bx],0h
;
;               write the record to selected record number
                mov     cl,writer
                mov     dx,offset fcb
                call    bdos
                or      al,al                   ;error code zero?
                jz      ready   ;for another record
                jmps     error   ;message if not
;
;
;
;  end of write command, process read
;
;
notw:
;               not a write command, read record?
                cmp     al,'R'
                jz      ranread
                jmps     error   ;skip if not
;
;               read random record
ranread:
                mov     cl,readr
                mov     dx,offset fcb
                call    bdos
                or      al,al                   ;return code 00?
                jz      readok
                jmps     error
;
;               read was successful, write to console
readok:
                call    crlf                    ;new line
                mov     cx,128                  ;max 128 characters
                mov     si,offset buff          ;next to get
wloop:
                lods    al                      ;next character
                and     al,07fh                 ;mask parity
                jnz     wloop1
                jmp     ready                   ;for another command if
wloop1:
                push    cx                      ;save counter
                push    si                      ;save next to get
                cmp     al,' '                  ;graphic?
                jb      skipw                   ;skip output if not grap
                call    putchr                  ;output character
skipw:
                pop     si
```

```
                pop     cx
                loop    wloop           ;decrement CX and check
                jmp     ready
;
;
;  end of read command, all errors end-up here
;
;
error:
                mov     dx,offset errmsg
                call    print
                jmp     ready
;
;  BDOS entry subroutine
bdos:
                int     224             ;entry to BDOS if by INT
                ret
;
abort:                                  ;return to CCP
                mov     cl,0
                call    bdos            ;use function 0 to end e
;
;  utility subroutines for console i/o
;
getchr:
                ;read next console character to a
                mov     cl,coninp
                call    bdos
                ret
;
putchr:
                ;write character from a to console
                mov     cl,conout
                mov     dl,al           ;character to send
                call    bdos            ;send character
                ret
;
crlf:
                ;send carriage return line feed
                mov     al,cr           ;carriage return
                call    putchr
                mov     al,lf           ;line feed
                call    putchr
                ret
;
print:
                ;print the buffer addressed by dx until $
                push    dx
                call    crlf
                pop     dx              ;new line
                mov     cl,pstring
                call    bdos            ;print the string
                ret
;
readcom:
```

```
        ;read the next command line to the conbuf
        mov     dx,offset prompt
        call    print               ;command?
        mov     cl,rstring
        mov     dx,offset conbuf
        call    bdos                ;read command line
;       command line is present, scan it
        mov     ax,0                ;start with 0000
        mov     bx,offset conlin
readc:  mov     dl,[bx]             ;next command character
        inc     bx                  ;to next command positio
        mov     dh,0                ;zero high byte for add
        or      dl,dl               ;check for end of comman
        jnz     getnum
        ret
;       not zero, numeric?
getnum:
        sub     dl,'0'
        cmp     dl,10               ;carry if numeric
        jnb     endrd
        mov     cl,10
        mul     cl                  ;multipy accumulator by
        add     ax,dx               ;+digit
        jmps     readc              ;for another char
endrd:
;       end of read, restore value in a and return value
        mov     dx,ax               ;return value  in DX
        mov     al,-1[bx]
        cmp     al,'a'              ;check for lower case
        jnb     transl
        ret
transl: and     al,5fH  ;translate to upper case
        ret
;
;
; Template for Page 0 of Data Group
;   Contains default FCB and DMA buffer
;
        dseg
        org     05ch
fcb     rb      33                  ;default file control bl
ranrec  rw      1                   ;random record position
ranovf  rb      1                   ;high order (overflow) b
buff    rb      128                 ;default DMA buffer
;
;  string data area for console messages
badver          db      'sorry, you need cp/m version 2$'
nospace         db      'no directory space$'
datmsg          db      'type data: $'
errmsg          db      'error, try again.$'
prompt          db      'next command? $'
;
;
;     fixed and variable data area
;
```

```
conbuf db       conlen  ;length of console buffer
consiz rs       1       ;resulting size after read
conlin rs       32      ;length 32 buffer
conlen equ      offset $ - offset consiz
;
            rs       31       ;16 level stack
stack       rb       1
            db       0        ;end byte for GENCMD
            end
```

Appendix C
Listing of the Boot ROM

```
***********************************************************
*                                                         *
* This is the original BOOT ROM distributed with CP/M    *
* for the SBC 86/12 and 204 Controller.  The listing     *
* is truncated on the right, but can be reproduced by    *
* assembling ROM.A86 from the distribution disk.  Note   *
* that the distributed source file should always be      *
* referenced for the latest version                      *
*                                                         *
***********************************************************
```

```
;
; ROM bootstrap for CP/M-86 on an iSBC86/12
;                with the
;   Intel SBC 204 Floppy Disk Controller
;
;       Copyright (C) 1980,1981
;       Digital Research, Inc.
;       Box 579, Pacific Grove
;       California, 93950
;
;*********************************************
;* This is the BOOT ROM which is initiated  *
;* by a system reset.  First, the ROM moves *
;* a copy of its data area to RAM at loca-  *
;* tion 00000H, then initializes the segment*
;* registers and the stack pointer.  The    *
;* various peripheral interface chips on the*
;* SBC 86/12 are initialized.  The 8251     *
;* serial interface is configured for a 9600*
;* baud asynchronous terminal, and the in-  *
;* terrupt controller is setup for inter-   *
;* rupts 10H-17H (vectors at 00040H-0005FH) *
;* and edge-triggered auto-EOI (end of in-  *
;* terrupt) mode with all interrupt levels  *
;* masked-off.  Next, the SBC 204 Diskette  *
;* controller is initialized, and track 1   *
;* sector 1 is read to determine the target *
;* paragraph address for LOADER.  Finally,  *
;* the LOADER on track 0 sectors 2-26 and   *
;* track 1 sectors 1-26 is read into the    *
;* target address.  Control then transfers  *
;* to LOADER.  This program resides in two  *
;* 2716 EPROM's (2K each) at location       *
;* 0FF000H on the SBC 86/12 CPU board.  ROM *
;* 0 contains the even memory locations, and*
;* ROM 1 contains the odd addresses.  BOOT  *
;* ROM uses RAM between 00000H and 000FFH   *
;* (absolute) for a scratch area, along with*
;* the sector 1 buffer.                     *
;*********************************************
```

```
00FF            true            equ     0ffh
FF00            false           equ     not true
                ;
00FF            debug           equ     true
                ;debug = true indicates bootstrap is in same roms
                ;with SBC 957 "Execution Vehicle" monitor
                ;at FE00:0 instead of FF00:0
                ;
000D            cr              equ     13
000A            lf              equ     10
                ;
                ;       disk ports and commands
                ;
00A0            base204         equ     0a0h
00A0            fdccom          equ     base204+0
00A0            fdcstat         equ     base204+0
00A1            fdcparm         equ     base204+1
00A1            fdcrslt         equ     base204+1
00A2            fdcrst          equ     base204+2
00A4            dmacadr         equ     base204+4
00A5            dmaccont        equ     base204+5
00A6            dmacscan        equ     base204+6
00A7            dmacsadr        equ     base204+7
00A8            dmacmode        equ     base204+8
00A8            dmacstat        equ     base204+8
00A9            fdcsel          equ     base204+9
00AA            fdcsegment      equ     base204+10
00AF            reset204        equ     base204+15
                ;
                ;actual console baud rate
2580            baud_rate       equ     9600
                ;value for 8253 baud counter
0008            baud            equ     768/(baud_rate/100)
                ;
00DA            csts            equ     0DAh    ;i8251 status port
00D8            cdata           equ     0D8h    ; "     data port
                ;
00D0            tch0            equ     0D0h    ;8253 PIC channel 0
00D2            tch1            equ     tch0+2  ;ch 1 port
00D4            tch2            equ     tch0+4  ;ch 2 port
00D6            tcmd            equ     tch0+6  ;8253 command port
                ;
00C0            icp1            equ     0C0h    ;8259a port 0
00C2            icp2            equ     0C2h    ;8259a port 1
                ;
                ;
                        IF NOT DEBUG
                ROMSEG          EQU     0FF00H  ;normal
                        ENDIF
                ;
                        IF DEBUG                ;share prom with SB
FE00            ROMSEG          EQU     0FE00H
                        ENDIF
                ;
                ;
```

```
                      ;       This long jump prom'd in by hand
                      ;       cseg    0ffffh          ;reset goes to here
                      ;       JMPF    BOTTOM          ;boot is at bottom
                      ;       EA 00 00 00 FF          ;cs = bottom of pro
                      ;                                        ip = 0
                      ;       EVEN PROM       ODD PROM
                      ;       7F8 - EA        7F8 - 00
                      ;       7F9 - 00        7F9 - 00
                      ;       7FA - FF                ;this is not done i
                      ;
 FE00                         cseg    romseg
                      ;
                      ;First, move our data area into RAM at 0000:0200
                      ;
0000 8CC8                     mov ax,cs
0002 8ED8                     mov ds,ax       ;point DS to CS for source
0004 BE3F01                   mov SI,drombegin        ;start of data
0007 BF0002                   mov DI,offset ram_start ;offset of destinat
000A B80000                   mov ax,0
000D 8EC0                     mov es,ax       ;destination segment is 000
000F B9E600                   mov CX,data_length      ;how much to move i
0012 F3A4                     rep movs al,al          ;move out of eprom
                      ;
0014 B80000                   mov ax,0
0017 8ED8                     mov ds,ax       ;data segment now in RAM
0019 8ED0                     mov ss,ax
001B BC2A03                   mov sp,stack_offset     ;Initialize stack s
001E FC                       cld                     ;clear the directio
                      ;
                              IF NOT DEBUG
                      ;
                      ;Now, initialize the console USART and baud rate
                      ;
                              mov al,0Eh
                              out csts,al     ;give 8251 dummy mode
                              mov al,40h
                              out csts,al     ;reset 8251 to accept mode
                              mov al,4Eh
                              out csts,al     ;normal 8 bit asynch mode,
                              mov al,37h
                              out csts,al     ;enable Tx & Rx
                              mov al,0B6h
                              out tcmd,al     ;8253 ch.2 square wave mode
                              mov ax,baud
                              out tch2,al     ;low of the baud rate
                              mov al,ah
                              out tch2,al     ;high of the baud rate
                      ;
                              ENDIF
                      ;
                      ;Setup the 8259 Programmable Interrupt Controller
                      ;
001F B013                     mov al,13h
0021 E6C0                     out icp1,al     ;8259a ICW 1  8086 mode
0023 B010                     mov al,10h
```

```
0025 E6C2                   out icp2,al     ;8259a ICW 2  vector @ 40-5
0027 B01F                   mov al,1Fh
0029 E6C2                   out icp2,al     ;8259a ICW 4  auto EOI mast
002B B0FF                   mov al,0FFh
002D E6C2                   out icp2,al     ;8259a OCW 1  mask all leve
                   ;
                   ;Reset and initialize the iSBC 204 Diskette Interfa
                   ;
                   restart:         ;also come back here on fatal error
002F E6AF                   out reset204,AL ;reset iSBC 204 logic and
0031 B001                   mov AL,1
0033 E6A2                   out fdcrst,AL   ;give 8271 FDC
0035 B000                   mov al,0
0037 E6A2                   out fdcrst,AL   ;  a reset command
0039 BB1502                 mov BX,offset specs1
003C E8E100                 CALL sendcom  ;program
003F BB1B02                 mov BX,offset specs2
0042 E8DB00                 CALL sendcom  ;  Shugart SA-800 drive
0045 BB2102                 mov BX,offset specs3
0048 E8D500                 call sendcom  ;    characteristics
004B BB1002        homer:   mov BX,offset home
004E E85800                 CALL execute    ;home drive 0
                   ;
0051 BB2A03                 mov bx,sector1  ;offset for first sector DM
0054 B80000                 mov ax,0
0057 8EC0                   mov es,ax       ;segment "    "    "    "
0059 E8A700                 call setup_dma
                   ;
005C BB0202                 mov bx,offset read0
005F E84700                 call execute    ;get T0 S1
                   ;
0062 8E062D03               mov es,ABS
0066 BB0000                 mov bx,0        ;get loader load address
0069 E89700                 call setup_dma  ;setup DMA to read loader
                   ;
006C BB0602                 mov bx,offset read1
006F E83700                 call execute    ;read track 0
0072 BB0B02                 mov bx,offset read2
0075 E83100                 call execute    ;read track 1
                   ;
0078 8C06E802               mov leap_segment,ES
                   ;        setup far jump vector
007C C706E6020000           mov leap_offset,0
                   ;
                   ;        enter LOADER
0082 FF2EE602               jmpf dword ptr leap_offset
                   ;
                   pmsg:
0086 8A0F                   mov cl,[BX]
0088 84C9                   test cl,cl
008A 7476                   jz return
008C E80400                 call conout
008F 43                     inc BX
0090 E9F3FF                 jmp pmsg
                   ;
```

```
                        conout:
0093 E4DA                       in al,csts
0095 A801                       test al,1
0097 74FA                       jz conout
0099 8AC1                       mov al,cl
009B E6D8                       out cdata,al
009D C3                         ret
                        ;
                        conin:
009E E4DA                       in al,csts
00A0 A802                       test al,2
00A2 74FA                       jz conin
00A4 E4D8                       in al,cdata
00A6 247F                       and al,7Fh
00A8 C3                         ret
                        ;
                        ;
                        ;
                        execute:        ;execute command string @ [BX]
                                        ;<BX> points to length,
                                        ;followed by Command byte
                                        ;followed by length-1 parameter byt
                        ;
00A9 891E0002                   mov     lastcom,BX      ;remember what it w
                        retry:                          ;retry if not ready
00AD E87000                     call    sendcom         ;execute the comman
                                                        ;now, let's see wha
                                                        ;of status poll was
                                                        ;for that command t
00B0 8B1E0002                   mov     BX,lastcom      ;point to command s
00B4 8A4701                     mov     AL,1[BX]        ;get command op cod
00B7 243F                       and     AL,3fh          ;drop drive code bi
00B9 B90008                     mov     CX,0800h        ;mask if it will be
00BC 3C2C                       cmp     AL,2ch          ;see if interrupt t
00BE 720B                       jb      execpoll
00C0 B98080                     mov     CX,8080h        ;else we use "not c
00C3 240F                       and     AL,0fh          ;unless . . .
00C5 3C0C                       cmp     AL,0ch          ;there isn't
00C7 B000                       mov AL,0
00C9 7737                       ja return               ;any result at all
                        ;
                        execpoll:       ;poll for bit in b, toggled with c
00CB E4A0                       in AL,FDCSTAT
00CD 22C5                       and AL,CH
00CF 32C174F8                   xor AL,CL ! JZ execpoll
                        ;
00D3 E4A1                       in      AL,fdcrslt      ;get result registe
00D5 241E                       and     AL,leh          ;look only at resul
00D7 7429                       jz      return          ;zero means it was
                        ;
00D9 3C10                       cmp al,10h
00DB 7513                       jne fatal               ;if other than "Not
                        ;
00DD BB1302                     mov bx,offset rdstat
00E0 E83D00                     call sendcom            ;perform read statu
```

```
                        rd_poll:
00E3 E4A0                       in al,fdc_stat
00E5 A880                       test al,80h             ;wait for command n
00E7 75FA                       jnz rd_poll
00E9 8B1E0002                   mov bx,last_com         ;recover last attem
00ED E9BDFF                     jmp retry               ;and try it over ag
                        ;
                        fatal:                          ; fatal error
00F0 B400                       mov ah,0
00F2 8BD8                       mov bx,ax               ;make 16 bits
00F4 8B9F2702                   mov bx,errtbl[BX]
                        ;       print appropriate error message
00F8 E88BFF                     call pmsg
00FB E8A0FF                     call conin              ;wait for key strik
00FE 58                         pop ax                  ;discard unused ite
00FF E92DFF                     jmp restart             ;then start all ove
                        ;
                        return:
0102 C3                         RET                     ;return from EXECUT
                        ;
                        setupdma:
0103 B004                       mov AL,04h
0105 E6A8                       out dmacmode,AL         ;enable dmac
0107 B000                       mov al,0
0109 E6A5                       out dmaccont,AL         ;set first (dummy)
010B B040                       mov AL,40h
010D E6A5                       out dmaccont,AL         ;force read data mo
010F 8CC0                       mov AX,ES
0111 E6AA                       out fdcsegment,AL
0113 8AC4                       mov AL,AH
0115 E6AA                       out fdcsegment,AL
0117 8BC3                       mov AX,BX
0119 E6A4                       out dmacadr,AL
011B 8AC4                       mov AL,AH
011D E6A4                       out dmacadr,AL
011F C3                         RET
                        ;
                        ;
                        ;
                        sendcom:        ;routine to send a command string t
0120 E4A0                       in AL,fdcstat
0122 2480                       and AL,80h
0124 75FA                       jnz sendcom     ;insure command not busy
0126 8A0F                       mov CL,[BX]     ;get count
0128 43                         inc BX
0129 8A07                       mov al,[BX]     ;point to and fetch command
012B E6A0                       out fdccom,AL   ;send command
                        parmloop:
012D FEC9                       dec CL
012F 74D1                       jz return       ;see if any (more) paramete
0131 43                         inc BX          ;point to next parameter
                        parmpoll:
0132 E4A0                       in AL,fdcstat
0134 2420                       and AL,20h
0136 75FA                       jnz parmpoll    ;loop until parm not full
```

```
0138 8A07                         mov AL,[BX]
013A E6A1                         out fdcparm,AL  ;output next parameter
013C E9EEFF                       jmp parmloop    ;go see about another
                    ;
                    ;
                    ;       Image of data to be moved to RAM
                    ;
  013F              drombegin equ offset $
                    ;
013F 0000           clastcom        dw      0000h     ;last command
                    ;
0141 03             creadstring     db      3         ;length
0142 52                             db      52h       ;read function code
0143 00                             db      0         ;track #
0144 01                             db      1         ;sector #
                    ;
0145 04             creadtrk0       db      4
0146 53                             db      53h       ;read multiple
0147 00                             db      0         ;track 0
0148 02                             db      2         ;sectors 2
0149 19                             db      25        ;through 26
                    ;
014A 04             creadtrk1       db      4
014B 53                             db      53h
014C 01                             db      1         ;track 1
014D 01                             db      1         ;sectors 1
014E 1A                             db      26        ;through 26
                    ;
014F 026900         chome0          db      2,69h,0
0152 016C           crdstat0        db      1,6ch
0154 05350D         cspecs1         db      5,35h,0dh
0157 0808E9                         db      08h,08h,0e9h
015A 053510         cspecs2         db      5,35h,10h
015D FFFFFF                         db      255,255,255
0160 053518         cspecs3         db      5,35h,18h
0163 FFFFFF                         db      255,255,255
                    ;
0166 4702           cerrtbl dw      offset er0
0168 4702                   dw      offset er1
016A 4702                   dw      offset er2
016C 4702                   dw      offset er3
016E 5702                   dw      offset er4
0170 6502                   dw      offset er5
0172 7002                   dw      offset er6
0174 7F02                   dw      offset er7
0176 9002                   dw      offset er8
0178 A202                   dw      offset er9
017A B202                   dw      offset erA
017C C502                   dw      offset erB
017E D302                   dw      offset erC
0180 4702                   dw      offset erD
0182 4702                   dw      offset erE
0184 4702                   dw      offset erF
                    ;
0186 0D0A4E756C6C Cer0     db      cr,lf,'Null Error ??',0
```

```
     204572726F72
     203F3F00
  0186              Cerl     equ     cer0
  0186              Cer2     equ     cer0
  0186              Cer3     equ     cer0
0196 0D0A436C6F63 Cer4     db      cr,lf,'Clock Error',0
     6B204572726F
     7200
01A4 0D0A4C617465 Cer5     db      cr,lf,'Late DMA',0
     20444D4100
01AF 0D0A49442043 Cer6     db      cr,lf,'ID CRC Error',0
     524320457272
     6F7200
01BE 0D0A44617461 Cer7     db      cr,lf,'Data CRC Error',0
     204352432045
     72726F7200
01CF 0D0A44726976 Cer8     db      cr,lf,'Drive Not Ready',0
     65204E6F7420
     526561647900
01E1 0D0A57726974 Cer9     db      cr,lf,'Write Protect',0
     652050726F74
     65637400
01F1 0D0A54726B20 CerA     db      cr,lf,'Trk 00 Not Found',0
     3030204E6F74
     20466F756E64
     00
0204 0D0A57726974 CerB     db      cr,lf,'Write Fault',0
     65204661756C
     7400
0212 0D0A53656374 CerC     db      cr,lf,'Sector Not Found',0
     6F72204E6F74
     20466F756E64
     00
  0186              CerD     equ     cer0
  0186              CerE     equ     cer0
  0186              CerF     equ     cer0
                    ;
  0225              dromend equ offset $
                    ;
  00E6              data_length      equ dromend-drombegin
                    ;
                    ;        reserve space in RAM for data area
                    ;        (no hex records generated here)
                    ;
  0000                       dseg    0
                             org     0200h
                    ;
  0200              ram_start        equ     $
0200                lastcom          rw      1        ;last command
0202                read0            rb      4        ;read track 0 secto
0206                read1            rb      5        ;read T0 S2-26
020B                read2            rb      5        ;read T1 S1-26
0210                home             rb      3        ;home drive 0
0213                rdstat           rb      2        ;read status
0215                specsl           rb      6
```

```
021B            specs2          rb      6
0221            specs3          rb      6
0227            errtbl          rw      16
0247            er0             rb      length cer0     ;16
  0247          er1             equ     er0
  0247          er2             equ     er0
  0247          er3             equ     er0
0257            er4             rb      length cer4     ;14
0265            er5             rb      length cer5     ;11
0270            er6             rb      length cer6     ;15
027F            er7             rb      length cer7     ;17
0290            er8             rb      length cer8     ;18
02A2            er9             rb      length cer9     ;16
02B2            erA             rb      length cerA     ;19
02C5            erB             rb      length cerB     ;14
02D3            erC             rb      length cerC     ;19
  0247          erD             equ     er0
  0247          erE             equ     er0
  0247          erF             equ     er0
                ;
02E6            leap_offset     rw      1
02E8            leap_segment    rw      1
                ;
                ;
02EA                            rw      32      ;local stack
  032A          stack_offset    equ     offset $;stack from here do
                ;
                ;               TO S1 read in here
  032A          sector1         equ offset $
                ;
032A            Ty              rb      1
032B            Len             rw      1
032D            Abs             rw      1       ;ABS is all we care
032F            Min             rw      1
0331            Max             rw      1
                                end
```

Appendix D
LDBIOS Listing

```
**************************************************
*                                                *
* This the the LOADER BIOS, derived from the BIOS *
* program by enabling the "loader_bios" condi-    *
* tional assembly switch.  The listing has been   *
* edited to remove portions which are duplicated  *
* in the BIOS listing which appears in Appendix D *
* where elipses "..." denote the deleted portions *
* (the listing is truncated on the right, but can *
* be reproduced by assembling the BIOS.A86 file   *
* provided with CP/M-86)                          *
*                                                *
**************************************************
```

```
                ;*********************************************
                ;*                                           *
                ;* Basic Input/Output System (BIOS) for      *
                ;* CP/M-86 Configured for iSBC 86/12 with    *
                ;* the iSBC 204 Floppy Disk Controller       *
                ;*                                           *
                ;* (Note:  this file contains both embedded  *
                ;* tabs and blanks to minimize the list file *
                ;* width for printing purposes.  You may wish*
                ;* to expand the blanks before performing    *
                ;* major editing.)                           *
                ;*********************************************

                ;       Copyright (C) 1980,1981
                ;       Digital Research, Inc.
                ;       Box 579, Pacific Grove
                ;       California, 93950
                ;
                ;       (Permission is hereby granted to use
                ;       or abstract the following program in
                ;       the implementation of CP/M, MP/M or
                ;       CP/NET for the 8086 or 8088 Micro-
                ;       processor)

 FFFF           true            equ -1
 0000           false           equ not true
```

```
                    ;*********************************************
                    ;*                                           *
                    ;* Loader_bios is true if assembling the     *
                    ;* LOADER BIOS, otherwise BIOS is for the    *
                    ;* CPM.SYS file.  Blc_list is true if we     *
                    ;* have a serial printer attached to BLC8538 *
                    ;* Bdos_int is interrupt used for earlier    *
                    ;* versions.                                 *
                    ;*                                           *
                    ;*********************************************

 FFFF               loader_bios     equ true
 FFFF               blc_list        equ true
 00E0               bdos_int        equ 224 ;reserved BDOS Interrupt

                            IF      not loader_bios
                    ;---------------------------------------------
                    ;|                                           |
                            . . .
                    ;|                                           |
                    ;---------------------------------------------
                            ENDIF   ;not loader_bios

                            IF      loader_bios
                    ;---------------------------------------------
                    ;|                                           |
 1200               bios_code       equ 1200h ;start of LDBIOS
 0003               ccp_offset      equ 0003h ;base of CPMLOADER
 0406               bdos_ofst       equ 0406h ;stripped BDOS entry
                    ;|                                           |
                    ;---------------------------------------------
                            ENDIF   ;loader_bios
                            . . .

                            cseg
                            org     ccpoffset
                    ccp:
                            org     bios_code

                    ;*********************************************
                    ;*                                           *
                    ;* BIOS Jump Vector for Individual Routines  *
                    ;*                                           *
                    ;*********************************************

1200 E93C00          jmp INIT        ;Enter from BOOT ROM or LOADER
1203 E96100          jmp WBOOT       ;Arrive here from BDOS call 0
                            . . .
1239 E96400          jmp GETIOBF     ;return I/O map byte (IOBYTE)
123C E96400          jmp SETIOBF     ;set I/O map byte (IOBYTE)
```

```
                     ;*********************************************
                     ;*                                           *
                     ;* INIT Entry Point, Differs for LDBIOS and  *
                     ;* BIOS, according to "Loader_Bios" value    *
                     ;*                                           *
                     ;*********************************************

                     INIT:   ;print signon message and initialize hardwa
123F 8CC8                    mov ax,cs         ;we entered with a JMPF so
1241 8ED0                    mov ss,ax         ; CS: as the initial value
1243 8ED8                    mov ds,ax         ;        DS:,
1245 8EC0                    mov es,ax         ;        and ES:
                             ;use local stack during initialization
1247 BCA916                  mov sp,offset stkbase
124A FC                      cld               ;set forward direction

                             IF      not loader_bios
                     ;---------------------------------------------
                     ;|                                            |
                             ; This is a BIOS for the CPM.SYS file.
                     ;|      . . .                                 |
                     ;---------------------------------------------
                             ENDIF   ;not loader_bios

                             IF      loader_bios
                     ;---------------------------------------------
                     ;|                                            |
                             ;This is a BIOS for the LOADER
124B 1E                      push ds           ;save data segment
124C B80000                  mov ax,0
124F 8ED8                    mov ds,ax         ;point to segment zero
                             ;BDOS interrupt offset
1251 C70680030604            mov bdos_offset,bdos_ofst
1257 8C0E8203                mov bdos_segment,CS ;bdos interrupt segment
125B 1F                      pop ds            ;restore data segment
                     ;|                                            |
                     ;---------------------------------------------
                             ENDIF   ;loader_bios

125C BB1514                  mov bx,offset signon
125F E85A00                  call pmsg         ;print signon message
1262 B100                    mov cl,0          ;default to dr A: on coldst
1264 E99CED                  jmp ccp           ;jump to cold start entry o

1267 E99FED          WBOOT:  jmp ccp+6         ;direct entry to CCP at com

                             IF      not loader_bios
                     ;---------------------------------------------
                     ;|                                            |
                             . . .
                     ;|                                            |
                     ;---------------------------------------------
                             ENDIF   ;not loader_bios
```

```
                ;*********************************************
                ;*                                           *
                ;*   CP/M Character I/O Interface Routines   *
                ;*   Console is Usart (i8251a) on iSBC 86/12 *
                ;*   at ports D8/DA                          *
                ;*                                           *
                ;*********************************************

                CONST:          ;console status
126A E4DA               in al,csts
                        . . .
                const_ret:
1272 C3                 ret             ;Receiver Data Available

                CONIN:                  ;console input
1273 E8F4FF             call const
                        . . .
                CONOUT:         ;console output
127D E4DA               in al,csts
                        . . .

                LISTOUT:                ;list device output

                        IF      blc_list
                ;---------------------------------------------
                ;|                                           |
1288 E80700             call LISTST
                          . . .
                ;|                                           |
                ;---------------------------------------------
                        ENDIF   ;blc_list

1291 C3                 ret

                LISTST:                 ;poll list status

                        IF      blc_list
                ;---------------------------------------------
                ;|                                           |
1292 E441               in al,lsts
                        . . .
                ;|                                           |
                ;---------------------------------------------
                        ENDIF   ;blc_list

129C C3                 ret

                PUNCH:  ;not implemented in this configuration
                READER:
129D B01A               mov al,lah
129F C3                 ret             ;return EOF for now
```

```
                        GETIOBF:
12A0 B000                       mov al,0        ;TTY: for consistency
12A2 C3                         ret             ;IOBYTE not implemented

                        SETIOBF:
12A3 C3                         ret             ;iobyte not implemented

                        zero_ret:
12A4 2400                       and al,0
12A6 C3                 ret                     ;return zero in AL and flag

                        ; Routine to get and echo a console character
                        ;       and shift it to upper case

                        uconecho:
12A7 E8C9FF                     call CONIN      ;get a console character
                                . . .
                        ;*********************************************
                        ;*                                           *
                        ;*          Disk Input/Output Routines       *
                        ;*                                           *
                        ;*********************************************

                        SELDSK:         ;select disk given by register CL
12CA BB0000                     mov bx,0000h
                                . . .

                        HOME:   ;move selected disk to home position (Track
12EB C606311500                 mov trk,0       ;set disk i/o to track zero
                                . . .

                        SETTRK: ;set track address given by CX
1300 880E3115                   mov trk,cl      ;we only use 8 bits of trac
1304 C3                         ret

                        SETSEC: ;set sector number given by cx
1305 880E3215                   mov sect,cl     ;we only use 8 bits of sect
1309 C3                         ret

                        SECTRAN: ;translate sector CX using table at [DX]
130A 8BD9                       mov bx,cx
                                . . .

                        SETDMA: ;set DMA offset given by CX
1311 890E2A15                   mov dma_adr,CX
1315 C3                         ret

                        SETDMAB: ;set DMA segment given by CX
1316 890E2C15                   mov dma_seg,CX
131A C3                         ret
                        ;
                        GETSEGT:  ;return address of physical memory table
131B BB3815                     mov bx,offset seg_table
131E C3                         ret
```

```
                  ;*********************************************
                  ;*                                           *
                  ;*  All disk I/O parameters are setup:  the  *
                  ;*  Read and Write entry points transfer one *
                  ;*  sector of 128 bytes to/from the current  *
                  ;*  DMA address using the current disk drive *
                  ;*                                           *
                  ;*********************************************

                  READ:
131F B012                 mov al,12h      ;basic read sector command
1321 EB02                 jmps r_w_common

                  WRITE:
1323 B00A                 mov al,0ah      ;basic write sector command

                  r_w_common:
1325 BB2F15               mov bx,offset io_com ;point to command stri
                          . . .

                  ;*********************************************
                  ;*                                           *
                  ;*               Data Areas                  *
                  ;*                                           *
                  ;*********************************************
  1415            data_offset     equ offset $

                          dseg
                          org     data_offset     ;contiguous with co

                          IF      loader_bios
                  ;--------------------------------------------
                  ;|                                          |
1415 0D0A0D0A     signon  db      cr,lf,cr,lf
1419 43502F4D2D38         db      'CP/M-86 Version 2.2',cr,lf,0
     362056657273
     696F6E20322E
     320D0A00
                  ;|                                          |
                  ;--------------------------------------------
                          ENDIF   ;loader_bios

                          IF      not loader_bios
                  ;--------------------------------------------
                  ;|                                          |
                          . . .
                  ;|                                          |
                  ;--------------------------------------------
                          ENDIF   ;not loader_bios

142F 0D0A486F6D65 bad_hom db      cr,lf,'Home Error',cr,lf,0
                          . . .
=                         include singles.lib ;read in disk definitio
=                 ;               DISKS 2
```

```
=  1541          dpbase  equ     $               ;Base of Disk Param
                         . . .
=1668 00                 db      0               ;Marks End of Modul

 1669            loc_stk rw  32  ;local stack for initialization
   16A9          stkbase equ offset $

                         . . .
 16A9 00                 db 0    ;fill last address for GENCMD

                 ;*********************************************
                 ;*                                           *
                 ;*          Dummy Data Section               *
                 ;*                                           *
                 ;*********************************************
   0000                  dseg    0       ;absolute low memory
                         org     0       ;(interrupt vectors)
                         . . .
                         END
```

Appendix E
BIOS Listing

```
*****************************************************
*                                                   *
* This is the CP/M-86 BIOS, derived from the BIOS *
* program by disabling the "loader_bios" condi-     *
* tional assembly switch.  The listing has been     *
* truncated on the right, but can be reproduced     *
* by assembling the BIOS.A86 file provided with     *
* CP/M-86.  This BIOS allows CP/M-86 operation      *
* with the Intel SBC 86/12 with the SBC 204 con-    *
* troller.  Use this BIOS, or the skeletal CBIOS    *
* listed in Appendix E, as the basis for a cus-     *
* tomized implementation of CP/M-86.                *
* provided with CP/M-86)                            *
*                                                   *
*****************************************************
```

```
                    ;*********************************************
                    ;*                                           *
                    ;* Basic Input/Output System (BIOS) for      *
                    ;* CP/M-86 Configured for iSBC 86/12 with    *
                    ;* the iSBC 204 Floppy Disk Controller       *
                    ;*                                           *
                    ;* (Note:  this file contains both embedded  *
                    ;* tabs and blanks to minimize the list file *
                    ;* width for printing purposes.  You may wish*
                    ;* to expand the blanks before performing    *
                    ;* major editing.)                           *
                    ;*********************************************

                    ;       Copyright (C) 1980,1981
                    ;       Digital Research, Inc.
                    ;       Box 579, Pacific Grove
                    ;       California, 93950
                    ;
                    ;       (Permission is hereby granted to use
                    ;       or abstract the following program in
                    ;       the implementation of CP/M, MP/M or
                    ;       CP/NET for the 8086 or 8088 Micro-
                    ;       processor)

   FFFF             true            equ -1
   0000             false           equ not true
```

```
                ;*********************************************
                ;*                                           *
                ;* Loader_bios is true if assembling the     *
                ;* LOADER_BIOS, otherwise BIOS is for the    *
                ;* CPM.SYS file.  Blc_list is true if we     *
                ;* have a serial printer attached to BLC8538 *
                ;* Bdos_int is interrupt used for earlier    *
                ;* versions.                                 *
                ;*                                           *
                ;*********************************************

0000            loader_bios     equ false
FFFF            blc_list        equ true
00E0            bdos_int        equ 224 ;reserved BDOS Interrupt

                        IF      not loader_bios
                ;---------------------------------------------
                ;|                                           |
2500            bios_code       equ 2500h
0000            ccp_offset      equ 0000h
0B06            bdos_ofst       equ 0B06h ;BDOS entry point
                ;|                                           |
                ;---------------------------------------------
                        ENDIF   ;not loader_bios

                        IF      loader_bios
                ;---------------------------------------------
                ;|                                           |
                bios_code       equ 1200h ;start of LDBIOS
                ccp_offset      equ 0003h ;base of CPMLOADER
                bdos_ofst       equ 0406h ;stripped BDOS entry
                ;|                                           |
                ;---------------------------------------------
                        ENDIF   ;loader_bios

00DA            csts            equ 0DAh  ;i8251 status port
00D8            cdata           equ 0D8h  ;   "  data port

                        IF      blc_list
                ;---------------------------------------------
                ;|                                           |
0041            lsts            equ 41h ;2651 No. 0 on BLC8538 stat
0040            ldata           equ 40h ; "    "   "  "     "  data
0060            blc_reset       equ 60h ;reset selected USARTS on B
                ;|                                           |
                ;---------------------------------------------
                        ENDIF   ;blc_list

                ;*********************************************
                ;*                                           *
                ;*    Intel iSBC 204 Disk Controller Ports   *
                ;*                                           *
                ;*********************************************
```

```
00A0            base204         equ 0a0h        ;SBC204 assigned ad

00A0            fdc_com         equ base204+0   ;8271 FDC out comma
00A0            fdc_stat        equ base204+0   ;8271 in status
00A1            fdc_parm        equ base204+1   ;8271 out parameter
00A1            fdc_rslt        equ base204+1   ;8271 in result
00A2            fdc_rst         equ base204+2   ;8271 out reset
00A4            dmac_adr        equ base204+4   ;8257 DMA base addr
00A5            dmac_cont       equ base204+5   ;8257 out control
00A6            dmac_scan       equ base204+6   ;8257 out scan cont
00A7            dmac_sadr       equ base204+7   ;8257 out scan addr
00A8            dmac_mode       equ base204+8   ;8257 out mode
00A8            dmac_stat       equ base204+8   ;8257 in status
00A9            fdc_sel         equ base204+9   ;FDC select port (n
00AA            fdc_segment     equ base204+10  ;segment address re
00AF            reset_204       equ base204+15  ;reset entire inter

000A            max_retries     equ 10          ;max retries on dis
                                                ;before perm error
000D            cr              equ 0dh         ;carriage return
000A            lf              equ 0ah         ;line feed

                        cseg
                        org     ccpoffset
                ccp:
                        org     bios_code

                ;*********************************************
                ;*                                           *
                ;* BIOS Jump Vector for Individual Routines  *
                ;*                                           *
                ;*********************************************

2500 E93C00       jmp INIT      ;Enter from BOOT ROM or LOADER
2503 E98400       jmp WBOOT     ;Arrive here from BDOS call 0
2506 E99000       jmp CONST     ;return console keyboard status
2509 E99600       jmp CONIN     ;return console keyboard char
250C E99D00       jmp CONOUT    ;write char to console device
250F E9A500       jmp LISTOUT   ;write character to list device
2512 E9B700       jmp PUNCH     ;write character to punch device
2515 E9B400       jmp READER    ;return char from reader device
2518 E9FF00       jmp HOME      ;move to trk 00 on cur sel drive
251B E9DB00       jmp SELDSK    ;select disk for next rd/write
251E E90E01       jmp SETTRK    ;set track for next rd/write
2521 E91001       jmp SETSEC    ;set sector for next rd/write
2524 E91901       jmp SETDMA    ;set offset for user buff (DMA)
2527 E92401       jmp READ      ;read a 128 byte sector
252A E92501       jmp WRITE     ;write a 128 byte sector
252D E99100       jmp LISTST    ;return list status
2530 E90601       jmp SECTRAN   ;xlate logical->physical sector
2533 E90F01       jmp SETDMAB   ;set seg base for buff (DMA)
2536 E91101       jmp GETSEGT   ;return offset of Mem Desc Table
2539 E99300       jmp GETIOBF   ;return I/O map byte (IOBYTE)
253C E99300       jmp SETIOBF   ;set I/O map byte (IOBYTE)
```

```
                ;*********************************************
                ;*                                           *
                ;* INIT Entry Point, Differs for LDBIOS and  *
                ;* BIOS, according to "Loader_Bios" value    *
                ;*                                           *
                ;*********************************************

                INIT:   ;print signon message and initialize hardwa
253F 8CC8               mov ax,cs          ;we entered with a JMPF so
2541 8ED0               mov ss,ax          ; CS: as the initial value
2543 8ED8               mov ds,ax          ;       DS:,
2545 8EC0               mov es,ax          ;       and ES:
                        ;use local stack during initialization
2547 BCE429             mov sp,offset stkbase
254A FC                 cld                ;set forward direction

                        IF      not loader_bios
                ;---------------------------------------------
                ;|                                           |
                        ; This is a BIOS for the CPM.SYS file.
                        ; Setup all interrupt vectors in low
                        ; memory to address trap

254B 1E                 push ds            ;save the DS register
254C B80000             mov ax,0
254F 8ED8               mov ds,ax
2551 8EC0               mov es,ax          ;set ES and DS to zero
                        ;setup interrupt 0 to address trap routine
2553 C70600008D25       mov int0_offset,offset int_trap
2559 8C0E0200           mov int0_segment,CS
255D BF0400             mov di,4
2560 BE0000             mov si,0           ;then propagate
2563 B9FE01             mov cx,510         ;trap vector to
2566 F3A5               rep movs ax,ax     ;all 256 interrupts
                        ;BDOS offset to proper interrupt
2568 C7068003060B       mov bdos_offset,bdos_ofst
256E 1F                 pop ds             ;restore the DS register

                ;*********************************************
                ;*                                           *
                ;* National "BLC 8538" Channel 0 for a serial*
                ;* 9600 baud printer - this board uses 8 Sig-*
                ;* netics 2651 Usarts which have on-chip baud*
                ;* rate generators.                          *
                ;*                                           *
                ;*********************************************

256F B0FF               mov al,0FFh
2571 E660               out blc_reset,al ;reset all usarts on 8538
2573 B04E               mov al,4Eh
2575 E642               out ldata+2,al   ;set usart 0 in async 8 bit
2577 B03E               mov al,3Eh
2579 E642               out ldata+2,al   ;set usart 0 to 9600 baud
257B B037               mov al,37h
257D E643               out ldata+3,al   ;enable Tx/Rx, and set up R
```

```
                        ;|                                         |
                        ;------------------------------------------
                                ENDIF   ;not loader_bios

                                IF      loader_bios
                        ;------------------------------------------
                        ;|                                         |
                                ;This is a BIOS for the LOADER
                                push ds         ;save data segment
                                mov ax,0
                                mov ds,ax       ;point to segment zero
                                ;BDOS interrupt offset
                                mov bdos_offset,bdos_ofst
                                mov bdos_segment,CS ;bdos interrupt segment
                                pop ds          ;restore data segment
                        ;|                                         |
                        ;------------------------------------------
                                ENDIF   ;loader_bios

257F BB4427                     mov bx,offset signon
2582 E86600                     call pmsg       ;print signon message
2585 B100                       mov cl,0        ;default to dr A: on coldst
2587 E976DA                     jmp ccp         ;jump to cold start entry o

258A E979DA             WBOOT:  jmp ccp+6       ;direct entry to CCP at com

                                IF      not loader_bios
                        ;------------------------------------------
                        ;|                                         |
                        int_trap:
258D FA                         cli             ;block interrupts
258E 8CC8                       mov ax,cs
2590 8ED8                       mov ds,ax       ;get our data segment
2592 BB7927                     mov bx,offset int_trp
2595 E85300                     call pmsg
2598 F4                         hlt             ;hardstop
                        ;|                                         |
                        ;------------------------------------------
                                ENDIF   ;not loader_bios

                        ;*********************************************
                        ;*                                           *
                        ;*   CP/M Character I/O Interface Routines   *
                        ;*   Console is Usart (i8251a) on iSBC 86/12 *
                        ;*   at ports D8/DA                          *
                        ;*                                           *
                        ;*********************************************

                        CONST:          ;console status
2599 E4DA                       in al,csts
259B 2402                       and al,2
259D 7402                       jz const_ret
259F 0CFF                       or al,255       ;return non-zero if RDA
                        const_ret:
25A1 C3                         ret             ;Receiver Data Available
```

```
                        CONIN:                        ;console input
25A2 E8F4FF                     call const
25A5 74FB                       jz CONIN              ;wait for RDA
25A7 E4D8                       in al,cdata
25A9 247F                       and al,7fh            ;read data and remove parit
25AB C3                         ret

                        CONOUT:          ;console output
25AC E4DA                       in al,csts
25AE 2401                       and al,1              ;get console status
25B0 74FA                       jz CONOUT             ;wait for TBE
25B2 8AC1                       mov al,cl
25B4 E6D8                       out cdata,al          ;Transmitter Buffer Empty
25B6 C3                         ret                   ;then return data

                        LISTOUT:                      ;list device output

                                IF       blc_list
                        ;-----------------------------------------------
                        ;|                                             |
25B7 E80700                     call LISTST
25BA 74FB                       jz LISTOUT            ;wait for printer not busy
25BC 8AC1                       mov al,cl
25BE E640                       out ldata,al          ;send char to TI 810
                        ;|                                             |
                        ;-----------------------------------------------
                                ENDIF    ;blc_list

25C0 C3                         ret

                        LISTST:                       ;poll list status

                                IF       blc_list
                        ;-----------------------------------------------
                        ;|                                             |
25C1 E441                       in al,lsts
25C3 2481                       and al,81h            ;look at both TxRDY and DTR
25C5 3C81                       cmp al,81h
25C7 750A                       jnz zero_ret          ;either false, printer is b
25C9 0CFF                       or al,255             ;both true, LPT is ready
                        ;|                                             |
                        ;-----------------------------------------------
                                ENDIF    ;blc_list

25CB C3                         ret

                        PUNCH:  ;not implemented in this configuration
                        READER:
25CC B01A                       mov al,1ah
25CE C3                         ret                   ;return EOF for now

                        GETIOBF:
25CF B000                       mov al,0              ;TTY: for consistency
25D1 C3                         ret                   ;IOBYTE not implemented
```

```
                    SETIOBF:
25D2 C3                     ret             ;iobyte not implemented

                    zero_ret:
25D3 2400                   and al,0
25D5 C3             ret                     ;return zero in AL and flag

                    ; Routine to get and echo a console character
                    ;       and shift it to upper case

                    uconecho:
25D6 E8C9FF                 call CONIN      ;get a console character
25D9 50                     push ax
25DA 8AC8                   mov cl,al       ;save and
25DC E8CDFF                 call CONOUT
25DF 58                     pop ax          ;echo to console
25E0 3C61                   cmp al,'a'
25E2 7206                   jb uret         ;less than 'a' is ok
25E4 3C7A                   cmp al,'z'
25E6 7702                   ja uret         ;greater than 'z' is ok
25E8 2C20                   sub al,'a'-'A'  ;else shift to caps
                    uret:
25EA C3                     ret

                    ;       utility subroutine to print messages

                    pmsg:
25EB 8A07                   mov al,[BX]     ;get next char from message
25ED 84C0                   test al,al
25EF 7428                   jz return       ;if zero return
25F1 8AC8                   mov CL,AL
25F3 E8B6FF                 call CONOUT     ;print it
25F6 43                     inc BX
25F7 EBF2                   jmps pmsg       ;next character and loop

                    ;*********************************************
                    ;*                                           *
                    ;*          Disk Input/Output Routines       *
                    ;*                                           *
                    ;*********************************************

                    SELDSK:         ;select disk given by register CL
25F9 BB0000                 mov bx,0000h
25FC 80F902                 cmp cl,2        ;this BIOS only supports 2
25FF 7318                   jnb return      ;return w/ 0000 in BX if ba
2601 B080                   mov al, 80h
2603 80F900                 cmp cl,0
2606 7502                   jne sell        ;drive 1 if not zero
2608 B040                   mov al, 40h     ;else drive is 0
260A A26928         sell:   mov sel_mask,al ;save drive select mask
                                            ;now, we need disk paramete
260D B500                   mov ch,0
260F 8BD9                   mov bx,cx       ;BX = word(CL)
2611 B104                   mov cl,4
```

```
2613 D3E3                       shl bx,cl       ;multiply drive code * 16
                                ;create offset from Disk Parameter Base
2615 81C37C28                   add bx,offset dp_base
                        return:
2619 C3                         ret

                        HOME:   ;move selected disk to home position (Track
261A C6066C2800                 mov trk,0       ;set disk i/o to track zero
261F BB6E28                     mov bx,offset hom_com
2622 E83500                     call execute
2625 74F2                       jz return       ;home drive and return if 0
2627 BB6A27                     mov bx,offset bad_hom   ;else print
262A E8BEFF                     call pmsg       ;"Home Error"
262D EBEB                       jmps home       ;and retry

                        SETTRK: ;set track address given by CX
262F 880E6C28                   mov trk,cl      ;we only use 8 bits of trac
2633 C3                         ret

                        SETSEC: ;set sector number given by cx
2634 880E6D28                   mov sect,cl     ;we only use 8 bits of sect
2638 C3                         ret

                        SECTRAN: ;translate sector CX using table at [DX]
2639 8BD9                       mov bx,cx
263B 03DA                       add bx,dx       ;add sector to tran table a
263D 8A1F                       mov bl,[bx]     ;get logical sector
263F C3                         ret

                        SETDMA: ;set DMA offset given by CX
2640 890E6528                   mov dma_adr,CX
2644 C3                         ret

                        SETDMAB: ;set DMA segment given by CX
2645 890E6728                   mov dma_seg,CX
2649 C3                         ret
                        ;
                        GETSEGT:  ;return address of physical memory table
264A BB7328                     mov bx,offset seg_table
264D C3                         ret

                        ;*********************************************
                        ;*                                           *
                        ;*  All disk I/O parameters are setup:  the  *
                        ;*  Read and Write entry points transfer one *
                        ;*  sector of 128 bytes to/from the current  *
                        ;*  DMA address using the current disk drive *
                        ;*                                           *
                        ;*********************************************

                        READ:
264E B012                       mov al,12h      ;basic read sector command
2650 EB02                       jmps r_w_common

                        WRITE:
```

```
2652 B00A                  mov al,0ah      ;basic write sector command

                  r_w_common:
2654 BB6A28                mov bx,offset io_com ;point to command stri
2657 884701                mov byte ptr 1[BX],al ;put command into str
                  ;        fall into execute and return

                  execute:  ;execute command string.
                           ;[BX] points to length,
                           ;       followed by Command byte,
                           ;       followed by length-1 parameter byte

265A 891E6328              mov last_com,BX ;save command address for r
                  outer_retry:
                           ;allow some retrying
265E C60662280A            mov rtry_cnt,max_retries
                  retry:
2663 8B1E6328              mov BX,last_com
2667 E88900                call send_com   ;transmit command to i8271
                  ;        check status poll

266A 8B1E6328              mov BX,last_com
266E 8A4701                mov al,1[bx]    ;get command op code
2671 B90008                mov cx,0800h    ;mask if it will be "int re
2674 3C2C                  cmp al,2ch
2676 720B                  jb exec_poll    ;ok if it is an interrupt t
2678 B98080                mov cx,8080h    ;else we use "not command b
267B 240F                  and al,0fh
267D 3C0C                  cmp al,0ch      ;unless there isn't
267F B000                  mov al,0
2681 7736                  ja exec_exit    ;       any result
                                           ;poll for bits in CH,
                  exec_poll:               ;  toggled with bits in CL

2683 E4A0                  in al,fdc_stat  ;read status
2685 22C5                  and al,ch
2687 32C1                  xor al,cl       ;  isolate what we want to
2689 74F8                  jz exec_poll    ;and loop until it is done

                                           ;Operation complete,
268B E4A1                  in al,fdc_rslt  ; see if result code indica
268D 241E                  and al,1eh
268F 7428                  jz exec_exit    ;no error, then exit
                                           ;some type of error occurre
2691 3C10                  cmp al,10h
2693 7425                  je dr_nrdy      ;was it a not ready drive ?
                                           ;no,
                  dr_rdy: ; then we just retry read or write
2695 FE0E6228              dec rtry_cnt
2699 75C8                  jnz retry       ;  up to 10 times

                  ;        retries do not recover from the
                  ;        hard error

269B B400                  mov ah,0
```

```
269D 8BD8                   mov bx,ax       ;make error code 16 bits
269F 8B9F9127               mov bx,errtbl[BX]
26A3 E845FF                 call pmsg       ;print appropriate message
26A6 E4D8                   in al,cdata     ;flush usart receiver buffe
26A8 E82BFF                 call uconecho   ;read upper case console ch
26AB 3C43                   cmp al,'C'
26AD 7425                   je wboot_1      ;cancel
26AF 3C52                   cmp al,'R'
26B1 74AB                   je outer_retry  ;retry 10 more times
26B3 3C49                   cmp al,'I'
26B5 741A                   je z_ret        ;ignore error
26B7 0CFF                   or al,255       ;set code for permanent err
                    exec_exit:
26B9 C3                     ret

                    dr_nrdy:        ;here to wait for drive ready
26BA E81A00                 call test_ready
26BD 75A4                   jnz retry       ;if it's ready now we are d
26BF E81500                 call test_ready
26C2 759F                   jnz retry       ;if not ready twice in row,
26C4 BB0228                 mov bx,offset nrdymsg
26C7 E821FF                 call pmsg ;"Drive Not Ready"
                    nrdy01:
26CA E80A00                 call test_ready
26CD 74FB                   jz nrdy01       ;now loop until drive ready
26CF EB92                   jmps retry      ;then go retry without decr
                    zret:
26D1 2400                   and al,0
26D3 C3                     ret             ;return with no error code

                    wboot_1:                ;can't make it w/ a short l
26D4 E9B3FE                 jmp WBOOT

                    ;*********************************************
                    ;*                                           *
                    ;*  The i8271 requires a read status command *
                    ;*  to reset a drive-not-ready after the     *
                    ;*  drive becomes ready                      *
                    ;*                                           *
                    ;*********************************************

                    test_ready:
26D7 B640                   mov dh, 40h     ;proper mask if dr 1
26D9 F606692880             test sel_mask,80h
26DE 7502                   jnz nrdy2
26E0 B604                   mov dh, 04h     ;mask for dr 0 status bit
                    nrdy2:
26E2 BB7128                 mov bx,offset rds_com
26E5 E80B00                 call send_com
                    dr_poll:
26E8 E4A0                   in al,fdc_stat  ;get status word
26EA A880                   test al,80h
26EC 75FA                   jnz dr_poll     ;wait for not command busy
26EE E4A1                   in al,fdc_rslt  ;get "special result"
26F0 84C6                   test al,dh      ;look at bit for this drive
```

```
26F2 C3                 ret             ;return status of ready

                ;*********************************************
                ;*                                           *
                ;*  Send_com sends a command and parameters  *
                ;*  to the i8271:  BX addresses parameters.  *
                ;*  The DMA controller is also initialized   *
                ;*  if this is a read or write               *
                ;*                                           *
                ;*********************************************

                send_com:
26F3 E4A0               in al,fdc_stat
26F5 A880               test al,80h     ;insure command not busy
26F7 75FA               jnz send_com    ;loop until ready

                        ;see if we have to initialize for a DMA ope

26F9 8A4701             mov al,1[bx]    ;get command byte
26FC 3C12               cmp al,12h
26FE 7504               jne write_maybe ;if not a read it could be
2700 B140               mov cl,40h
2702 EB06               jmps init_dma   ;is a read command, go set
                write_maybe:
2704 3C0A               cmp al,0ah
2706 7520               jne dma_exit    ;leave DMA alone if not rea
2708 B180               mov cl,80h      ;we have write, not read
                init_dma:
                ;we have a read or write operation, setup DMA contr
                ;       (CL contains proper direction bit)
270A B004               mov al,04h
270C E6A8               out dmac_mode,al    ;enable dmac
270E B000               mov al,00
2710 E6A5               out dmac_cont,al    ;send first byte to con
2712 8AC1               mov al,cl
2714 E6A5               out dmac_cont,al   ;load direction register
2716 A16528             mov ax,dma_adr
2719 E6A4               out dmac_adr,al    ;send low byte of DMA
271B 8AC4               mov al,ah
271D E6A4               out dmac_adr,al    ;send high byte
271F A16728             mov ax,dma_seg
2722 E6AA               out fdc_segment,al ;send low byte of segmen
2724 8AC4               mov al,ah
2726 E6AA               out fdc_segment,al ;then high segment addre
                dma_exit:
2728 8A0F               mov cl,[BX]     ;get count
272A 43                 inc BX
272B 8A07               mov al,[BX]     ;get command
272D 0A066928           or al,sel_mask  ;merge command and drive co
2731 E6A0               out fdc_com,al  ;send command byte
                parm_loop:
2733 FEC9               dec cl
2735 7482               jz exec_exit    ;no (more) parameters, retu
2737 43                 inc BX          ;point to (next) parameter
                parm_poll:
```

```
2738 E4A0                   in al,fdc_stat
273A A820                   test al,20h     ;test "parameter register f
273C 75FA                   jnz parm_poll   ;idle until parm req not fu
273E 8A07                   mov al,[BX]
2740 E6A1                   out fdc_parm,al ;send next parameter
2742 EBEF                   jmps parm_loop  ;go see if there are more p

                    ;**********************************************
                    ;*                                            *
                    ;*                Data Areas                  *
                    ;*                                            *
                    ;**********************************************
  2744              data_offset     equ offset $

                            dseg
                            org     data_offset     ;contiguous with co

                            IF      loader_bios
                    ;----------------------------------------------
                    ;|                                            |
                    signon  db      cr,lf,cr,lf
                            db      'CP/M-86 Version 2.2',cr,lf,0
                    ;|                                            |
                    ;----------------------------------------------
                            ENDIF   ;loader_bios

                            IF      not loader_bios
                    ;----------------------------------------------
                    ;|                                            |
2744 0D0A0D0A       signon  db      cr,lf,cr,lf
2748 202053797374           db      '  System Generated  - 11 Jan 81',c
     656D2047656E
     657261746564
     20202D203131
     204A616E2038
     310D0A00
                    ;|                                            |
                    ;----------------------------------------------
                            ENDIF   ;not loader_bios

276A 0D0A486F6D65 bad_hom db        cr,lf,'Home Error',cr,lf,0
     204572726F72
     0D0A00
2779 0D0A496E7465 int_trp db        cr,lf,'Interrupt Trap Halt',cr,lf,0
     727275707420
     547261702048
     616C740D0A00

2791 B127B127B127 errtbl  dw er0,er1,er2,er3
     B127
2799 C127D127DE27         dw er4,er5,er6,er7
     EF27
27A1 022816282828         dw er8,er9,erA,erB
     3D28
27A9 4D28B127B127         dw erC,erD,erE,erF
```

```
     B127

27B1 0D0A4E756C6C er0      db  cr,lf,'Null Error ??',0
     204572726F72
     203F3F00
  27B1             er1      equ er0
  27B1             er2      equ er0
  27B1             er3      equ er0
27C1 0D0A436C6F63 er4      db  cr,lf,'Clock Error :',0
     6B204572726F
     72203A00
27D1 0D0A4C617465 er5      db  cr,lf,'Late DMA :',0
     20444D41203A
     00
27DE 0D0A49442043 er6      db  cr,lf,'ID CRC Error :',0
     524320457272
     6F72203A00
27EF 0D0A44617461 er7      db  cr,lf,'Data CRC Error :',0
     204352432045
     72726F72203A
     00
2802 0D0A44726976 er8      db  cr,lf,'Drive Not Ready :',0
     65204E6F7420
     526561647920
     3A00
2816 0D0A57726974 er9      db  cr,lf,'Write Protect :',0
     652050726F74
     656374203A00
2828 0D0A54726B20 erA      db  cr,lf,'Trk 00 Not Found :',0
     3030204E6F74
     20466F756E64
     203A00
283D 0D0A57726974 erB      db  cr,lf,'Write Fault :',0
     65204661756C
     74203A00
284D 0D0A53656374 erC      db  cr,lf,'Sector Not Found :',0
     6F72204E6F74
     20466F756E64
     203A00
  27B1             erD      equ er0
  27B1             erE      equ er0
  27B1             erF      equ er0
  2802             nrdymsg  equ er8

2862 00            rtry_cnt db 0    ;disk error retry counter
2863 0000          last_com dw 0    ;address of last command string
2865 0000          dma_adr  dw 0    ;dma offset stored here
2867 0000          dma_seg  dw 0    ;dma segment stored here
2869 40            sel_mask db 40h  ;select mask, 40h or 80h

                   ;        Various command strings for i8271

286A 03            io_com   db 3    ;length
286B 00            rd_wr    db 0    ;read/write function code
286C 00            trk      db 0    ;track #
```

```
 286D 00              sect    db 0    ;sector #

 286E 022900          hom_com db 2,29h,0       ;home drive command
 2871 012C            rds_com db 1,2ch         ;read status command

                      ;       System Memory Segment Table

 2873 02              segtable db 2   ;2 segments
 2874 DF02                    dw tpa_seg       ;1st seg starts after BIOS
 2876 2105                    dw tpa_len       ;and extends to 08000
 2878 0020                    dw 2000h         ;second is 20000 -
 287A 0020                    dw 2000h         ;3FFFF (128k)

=                             include singles.lib ;read in disk definitio
=                     ;                DISKS 2
=   287C              dpbase  equ      $                   ;Base of Disk Param
=287C AB280000        dpe0    dw       xlt0,0000h          ;Translate Table
=2880 00000000                dw       0000h,0000h         ;Scratch Area
=2884 C5289C28                dw       dirbuf,dpb0         ;Dir Buff, Parm Blo
=2888 64294529                dw       csv0,alv0           ;Check, Alloc Vecto
=288C AB280000        dpe1    dw       xlt1,0000h          ;Translate Table
=2890 00000000                dw       0000h,0000h         ;Scratch Area
=2894 C5289C28                dw       dirbuf,dpb1         ;Dir Buff, Parm Blo
=2898 93297429                dw       csv1,alv1           ;Check, Alloc Vecto
=                     ;                DISKDEF 0,1,26,6,1024,243,64,64,2
=   289C              dpb0    equ      offset $            ;Disk Parameter Blo
=289C 1A00                    dw       26                  ;Sectors Per Track
=289E 03                      db       3                   ;Block Shift
=289F 07                      db       7                   ;Block Mask
=28A0 00                      db       0                   ;Extnt Mask
=28A1 F200                    dw       242                 ;Disk Size - 1
=28A3 3F00                    dw       63                  ;Directory Max
=28A5 C0                      db       192                 ;Alloc0
=28A6 00                      db       0                   ;Alloc1
=28A7 1000                    dw       16                  ;Check Size
=28A9 0200                    dw       2                   ;Offset
=   28AB              xlt0    equ      offset $            ;Translate Table
=28AB 01070D13                db       1,7,13,19
=28AF 19050B11                db       25,5,11,17
=28B3 1703090F                db       23,3,9,15
=28B7 1502080E                db       21,2,8,14
=28BB 141A060C                db       20,26,6,12
=28BF 1218040A                db       18,24,4,10
=28C3 1016                    db       16,22
=   001F              als0    equ      31                  ;Allocation Vector
=   0010              css0    equ      16                  ;Check Vector Size
=                     ;                DISKDEF 1,0
=   289C              dpb1    equ      dpb0                ;Equivalent Paramet
=   001F              als1    equ      als0                ;Same Allocation Ve
=   0010              css1    equ      css0                ;Same Checksum Vect
=   28AB              xlt1    equ      xlt0                ;Same Translate Tab
=                     ;                ENDEF
=                     ;
=                     ;       Uninitialized Scratch Memory Follows:
=   28C5              begdat  equ      offset $            ;Start of Scratch A
```

```
=28C5           dirbuf  rs      128             ;Directory Buffer
=2945           alv0    rs      als0            ;Alloc Vector
=2964           csv0    rs      css0            ;Check Vector
=2974           alv1    rs      als1            ;Alloc Vector
=2993           csv1    rs      css1            ;Check Vector
=  29A3         enddat  equ     offset $        ;End of Scratch Are
=  00DE         datsiz  equ     offset $-begdat ;Size of Scratch Ar
=29A3 00                db      0               ;Marks End of Modul

 29A4           loc_stk rw  32  ;local stack for initialization
   29E4         stkbase equ offset $

   29E4         lastoff equ offset $
   02DF         tpa_seg equ (lastoff+0400h+15) / 16
   0521         tpa_len equ 0800h - tpa_seg
 29E4 00                db 0    ;fill last address for GENCMD

                ;**********************************************
                ;*                                            *
                ;*           Dummy Data Section               *
                ;*                                            *
                ;**********************************************
  0000                  dseg    0       ;absolute low memory
                        org     0       ;(interrupt vectors)
 0000           int0_offset     rw      1
 0002           int0_segment    rw      1
                ;       pad to system call vector
 0004                   rw      2*(bdos_int-1)

 0380           bdos_offset     rw      1
 0382           bdos_segment    rw      1
                        END
```

Appendix F
CBIOS Listing

```
**************************************************
*                                                *
* This is the listing of the skeletal CBIOS which *
* you can use as the basis for a customized BIOS  *
* for non-standard hardware.  The essential por-  *
* tions of the BIOS remain, with "rs" statements  *
* marking the routines to be inserted.            *
*                                                *
**************************************************

                ;*********************************************
                ;*                                           *
                ;* This Customized BIOS adapts CP/M-86 to    *
                ;* the following hardware configuration      *
                ;*     Processor:                            *
                ;*     Brand:                                *
                ;*     Controller:                           *
                ;*                                           *
                ;*                                           *
                ;*     Programmer:                           *
                ;*     Revisions :                           *
                ;*                                           *
                ;*********************************************

FFFF            true            equ -1
0000            false           equ not true
000D            cr              equ 0dh ;carriage return
000A            lf              equ 0ah ;line feed

                ;*********************************************
                ;*                                           *
                ;* Loader_bios is true if assembling the     *
                ;* LOADER BIOS, otherwise BIOS is for the    *
                ;* CPM.SYS file.                             *
                ;*                                           *
                ;*********************************************

0000            loader_bios     equ false
00E0            bdos_int        equ 224 ;reserved BDOS interrupt

                        IF      not loader_bios
                ;---------------------------------------------
                ;|                                           |
2500            bios_code       equ 2500h
0000            ccp_offset      equ 0000h
0B06            bdos_ofst       equ 0B06h ;BDOS entry point
                ;|                                           |
                ;---------------------------------------------
```

```
                        ENDIF   ;not loader_bios

                        IF      loader_bios
                ;-----------------------------------------
                ;|                                       |
                bios_code       equ 1200h ;start of LDBIOS
                ccp_offset      equ 0003h ;base of CPMLOADER
                bdos_ofst       equ 0406h ;stripped BDOS entry
                ;|                                       |
                ;-----------------------------------------
                        ENDIF   ;loader_bios

                        cseg
                        org     ccpoffset
                ccp:
                        org     bios_code

                ;*****************************************
                ;*                                       *
                ;* BIOS Jump Vector for Individual Routines *
                ;*                                       *
                ;*****************************************

2500 E93C00             jmp INIT        ;Enter from BOOT ROM or LOADER
2503 E97900             jmp WBOOT       ;Arrive here from BDOS call 0
2506 E98500             jmp CONST       ;return console keyboard status
2509 E98D00             jmp CONIN       ;return console keyboard char
250C E99A00             jmp CONOUT      ;write char to console device
250F E9A200             jmp LISTOUT     ;write character to list device
2512 E9B500             jmp PUNCH       ;write character to punch device
2515 E9BD00             jmp READER      ;return char from reader device
2518 E9F600             jmp HOME        ;move to trk 00 on cur sel drive
251B E9D900             jmp SELDSK      ;select disk for next rd/write
251E E90101             jmp SETTRK      ;set track for next rd/write
2521 E90301             jmp SETSEC      ;set sector for next rd/write
2524 E90C01             jmp SETDMA      ;set offset for user buff (DMA)
2527 E91701             jmp READ        ;read a 128 byte sector
252A E94701             jmp WRITE       ;write a 128 byte sector
252D E98F00             jmp LISTST      ;return list status
2530 E9F900             jmp SECTRAN     ;xlate logical->physical sector
2533 E90201             jmp SETDMAB     ;set seg base for buff (DMA)
2536 E90401             jmp GETSEGT     ;return offset of Mem Desc Table
2539 E9A400             jmp GETIOBF     ;return I/O map byte (IOBYTE)
253C E9A500             jmp SETIOBF     ;set I/O map byte (IOBYTE)

                ;*****************************************
                ;*                                       *
                ;* INIT Entry Point, Differs for LDBIOS and *
                ;* BIOS, according to "Loader_Bios" value *
                ;*                                       *
                ;*****************************************

                INIT:   ;print signon message and initialize hardwa
253F 8CC8               mov ax,cs       ;we entered with a JMPF so
```

```
2541 8ED0                   mov ss,ax          ;CS: as the initial value o
2543 8ED8                   mov ds,ax          ;DS:,
2545 8EC0                   mov es,ax          ;and ES:
                            ;use local stack during initialization
2547 BC5928                 mov sp,offset stkbase
254A FC                     cld                ;set forward direction

                            IF      not loader_bios
                    ;---------------------------------------------
                    ;|                                           |
                            ; This is a BIOS for the CPM.SYS file.
                            ; Setup all interrupt vectors in low
                            ; memory to address trap

254B 1E                     push ds            ;save the DS register
254C C606A72600             mov IOBYTE,0       ;clear IOBYTE
2551 B80000                 mov ax,0
2554 8ED8                   mov ds,ax
2556 8EC0                   mov es,ax          ;set ES and DS to zero
                            ;setup interrupt 0 to address trap routine
2558 C70600008225           mov int0_offset,offset int_trap
255E 8C0E0200               mov int0_segment,CS
2562 BF0400                 mov di,4
2565 BE0000                 mov si,0           ;then propagate
2568 B9FE01                 mov cx,510         ;trap vector to
256B F3A5                   rep movs ax,ax     ;all 256 interrupts
                            ;BDOS offset to proper interrupt
256D C7068003060B           mov bdos_offset,bdos_ofst
2573 1F                     pop ds             ;restore the DS register

                    ;       (additional CP/M-86 initialization)
                    ;|                                           |
                    ;---------------------------------------------
                            ENDIF   ;not loader_bios

                            IF      loader_bios
                    ;---------------------------------------------
                    ;|                                           |
                            ;This is a BIOS for the LOADER
                            push ds            ;save data segment
                            mov ax,0
                            mov ds,ax          ;point to segment zero
                            ;BDOS interrupt offset
                            mov bdos_offset,bdos_ofst
                            mov bdos_segment,CS ;bdos interrupt segment
                    ;       (additional LOADER initialization)
                            pop ds             ;restore data segment
                    ;|                                           |
                    ;---------------------------------------------
                            ENDIF   ;loader_bios

2574 BBB126                 mov bx,offset signon
2577 E86F00                 call pmsg          ;print signon message
257A B100                   mov cl,0           ;default to dr A: on coldst
257C E981DA                 jmp ccp            ;jump to cold start entry o
```

```
257F E984DA         WBOOT:  jmp ccp+6       ;direct entry to CCP at com

                            IF      not loader_bios
                    ;--------------------------------------------
                    ;|                                          |
                    int_trap:
2582 FA                     cli             ;block interrupts
2583 8CC8                   mov ax,cs
2585 8ED8                   mov ds,ax       ;get our data segment
2587 BBD126                 mov bx,offset int_trp
258A E85C00                 call pmsg
258D F4                     hlt             ;hardstop
                    ;|                                          |
                    ;--------------------------------------------
                            ENDIF   ;not loader_bios

                    ;********************************************
                    ;*                                          *
                    ;*   CP/M Character I/O Interface Routines  *
                    ;*                                          *
                    ;********************************************

                    CONST:          ;console status
258E                        rs      10      ;(fill-in)
2598 C3                     ret

                    CONIN:                  ;console input
2599 E8F2FF                 call CONST
259C 74FB                   jz CONIN        ;wait for RDA
259E                        rs      10      ;(fill-in)
25A8 C3                     ret

                    CONOUT:         ;console output
25A9                        rs      10      ;(fill-in)
25B3 C3                     ret             ;then return data

                    LISTOUT:                ;list device output
25B4                        rs      10      ;(fill-in)
25BE C3                     ret

                    LISTST:                 ;poll list status
25BF                        rs      10      ;(fill-in)
25C9 C3                     ret

                    PUNCH:          ;write punch device
25CA                        rs      10      ;(fill-in)
25D4 C3                     ret

                    READER:
25D5                        rs      10      ;(fill-in)
25DF C3                     ret

                    GETIOBF:
25E0 A0A726                 mov al,IOBYTE
```

```
25E3 C3                         ret

                        SETIOBF:
25E4 880EA726                   mov IOBYTE,cl   ;set iobyte
25E8 C3                         ret             ;iobyte not implemented

                        pmsg:
25E9 8A07                       mov al,[BX]     ;get next char from message
25EB 84C0                       test al,al
25ED 7421                       jz return       ;if zero return
25EF 8AC8                       mov CL,AL
25F1 E8B5FF                     call CONOUT     ;print it
25F4 43                         inc BX
25F5 EBF2                       jmps pmsg       ;next character and loop

                        ;*********************************************
                        ;*                                           *
                        ;*          Disk Input/Output Routines       *
                        ;*                                           *
                        ;*********************************************

                        SELDSK:         ;select disk given by register CL
  0002                  ndisks  equ     2 ;number of disks (up to 16)
25F7 880EA826                   mov disk,cl     ;save disk number
25FB BB0000                     mov bx,0000h    ;ready for error return
25FE 80F902                     cmp cl,ndisks   ;n beyond max disks?
2601 730D                       jnb return      ;return if so
2603 B500                       mov ch,0        ;double(n)
2605 8BD9                       mov bx,cx       ;bx = n
2607 B104                       mov cl,4        ;ready for *16
2609 D3E3                       shl bx,cl       ;n = n * 16
260B B9F126                     mov cx,offset dpbase
260E 03D9                       add bx,cx       ;dpbase + n * 16
2610 C3                 return: ret             ;bx = .dph

                        HOME:   ;move selected disk to home position (Track
2611 C706A9260000               mov trk,0       ;set disk i/o to track zero
2617                            rs      10      ;(fill-in)
2621 C3                         ret

                        SETTRK: ;set track address given by CX
2622 890EA926                   mov trk,CX
2626 C3                         ret

                        SETSEC: ;set sector number given by cx
2627 890EAB26                   mov sect,CX
262B C3                         ret

                        SECTRAN: ;translate sector CX using table at [DX]
262C 8BD9                       mov bx,cx
262E 03DA                       add bx,dx       ;add sector to tran table a
2630 8A1F                       mov bl,[bx]     ;get logical sector
2632 C3                         ret

                        SETDMA: ;set DMA offset given by CX
```

```
2633 890EAD26               mov dma_adr,CX
2637 C3                     ret

                     SETDMAB: ;set DMA segment given by CX
2638 890EAF26               mov dma_seg,CX
263C C3                     ret
                     ;
                     GETSEGT:  ;return address of physical memory table
263D BBE826                 mov bx,offset seg_table
2640 C3                     ret

                     ;*********************************************
                     ;*                                           *
                     ;*  All disk I/O parameters are setup:       *
                     ;*     DISK     is disk number      (SELDSK) *
                     ;*     TRK      is track number     (SETTRK) *
                     ;*     SECT     is sector number    (SETSEC) *
                     ;*     DMA_ADR  is the DMA offset   (SETDMA) *
                     ;*     DMA_SEG  is the DMA segment  (SETDMAB)*
                     ;*  READ reads the selected sector to the DMA*
                     ;*  address, and WRITE writes the data from  *
                     ;*  the DMA address to the selected sector   *
                     ;*  (return 00 if successful,  01 if perm err)*
                     ;*                                           *
                     ;*********************************************

                     READ:
2641                        rs      50       ;fill-in
2673 C3                     ret

                     WRITE:
2674                        rs      50       ;(fill-in)
26A6 C3                     ret


                     ;*********************************************
                     ;*                                           *
                     ;*                 Data Areas                *
                     ;*                                           *
                     ;*********************************************
  26A7               data_offset     equ offset $

                            dseg
                            org     data_offset     ;contiguous with co
26A7 00              IOBYTE db      0
26A8 00              disk   db      0        ;disk number
26A9 0000            trk    dw      0        ;track number
26AB 0000            sect   dw      0        ;sector number
26AD 0000            dma_adr dw     0        ;DMA offset from DS
26AF 0000            dma_seg dw     0        ;DMA Base Segment

                            IF      loader_bios
                     ;---------------------------------------------
                     ;|                                           |
                     signon db      cr,lf,cr,lf
```

```
                            db      'CP/M-86 Version 1.0',cr,lf,0
                    ;|                                              |
                    ;-----------------------------------------------
                            ENDIF   ;loader_bios

                            IF      not loader_bios
                    ;-----------------------------------------------
                    ;|                                              |
26B1 0D0A0D0A       signon  db      cr,lf,cr,lf
26B5 53797374656D           db      'System Generated 00/00/00'
     2047656E6572
     617465642030
     302F30302F30
     30
26CE 0D0A00                 db      cr,lf,0
                    ;|                                              |
                    ;-----------------------------------------------
                            ENDIF   ;not loader_bios

26D1 0D0A           int_trp db      cr,lf
26D3 496E74657272           db      'Interrupt Trap Halt'
     757074205472
     61702048616C
     74
26E6 0D0A                   db      cr,lf

                    ;       System Memory Segment Table

26E8 02             segtable db 2   ;2 segments
26E9 C602                   dw tpa_seg      ;1st seg starts after BIOS
26EB 3A05                   dw tpa_len      ;and extends to 08000
26ED 0020                   dw 2000h        ;second is 20000 -
26EF 0020                   dw 2000h        ;3FFFF (128k)

=                           include singles.lib ;read in disk definitio
=                   ;               DISKS 2
=  26F1             dpbase  equ     $               ;Base of Disk Param
=26F1 20270000      dpe0    dw      xlt0,0000h      ;Translate Table
=26F5 00000000              dw      0000h,0000h     ;Scratch Area
=26F9 3A271127              dw      dirbuf,dpb0     ;Dir Buff, Parm Blo
=26FD D927BA27              dw      csv0,alv0       ;Check, Alloc Vecto
=2701 20270000      dpe1    dw      xlt1,0000h      ;Translate Table
=2705 00000000              dw      0000h,0000h     ;Scratch Area
=2709 3A271127              dw      dirbuf,dpb1     ;Dir Buff, Parm Blo
=270D 0828E927              dw      csv1,alv1       ;Check, Alloc Vecto
=                   ;               DISKDEF 0,1,26,6,1024,243,64,64,2
=  2711             dpb0    equ     offset $        ;Disk Parameter Blo
=2711 1A00                  dw      26              ;Sectors Per Track
=2713 03                    db      3               ;Block Shift
=2714 07                    db      7               ;Block Mask
=2715 00                    db      0               ;Extnt Mask
=2716 F200                  dw      242             ;Disk Size - 1
=2718 3F00                  dw      63              ;Directory Max
=271A C0                    db      192             ;Alloc0
=271B 00                    db      0               ;Alloc1
```

```
=271C 1000              dw      16              ;Check Size
=271E 0200              dw      2               ;Offset
=  2720         xlt0    equ     offset $        ;Translate Table
=2720 01070D13          db      1,7,13,19
=2724 19050B11          db      25,5,11,17
=2728 1703090F          db      23,3,9,15
=272C 1502080E          db      21,2,8,14
=2730 141A060C          db      20,26,6,12
=2734 1218040A          db      18,24,4,10
=2738 1016              db      16,22
=  001F         als0    equ     31              ;Allocation Vector
=  0010         css0    equ     16              ;Check Vector Size
=               ;               DISKDEF 1,0
=  2711         dpb1    equ     dpb0            ;Equivalent Paramet
=  001F         als1    equ     als0            ;Same Allocation Ve
=  0010         css1    equ     css0            ;Same Checksum Vect
=  2720         xlt1    equ     xlt0            ;Same Translate Tab
=               ;               ENDEF
=               ;
=               ;       Uninitialized Scratch Memory Follows:
=               ;
=  273A         begdat  equ     offset $        ;Start of Scratch A
=273A           dirbuf  rs      128             ;Directory Buffer
=27BA           alv0    rs      als0            ;Alloc Vector
=27D9           csv0    rs      css0            ;Check Vector
=27E9           alv1    rs      als1            ;Alloc Vector
=2808           csv1    rs      css1            ;Check Vector
=  2818         enddat  equ     offset $        ;End of Scratch Are
=  00DE         datsiz  equ     offset $-begdat ;Size of Scratch Ar
=2818 00                db      0               ;Marks End of Modul

 2819           loc_stk rw  32  ;local stack for initialization
   2859         stkbase equ offset $

   2859         lastoff equ offset $
   02C6         tpa_seg equ (lastoff+0400h+15) / 16
   053A         tpa_len equ 0800h - tpa_seq
 2859 00                db 0    ;fill last address for GENCMD

                ;*********************************************
                ;*                                           *
                ;*          Dummy Data Section               *
                ;*                                           *
                ;*********************************************
   0000                 dseg    0       ;absolute low memory
                        org     0       ;(interrupt vectors)
 0000           int0_offset     rw      1
 0002           int0_segment    rw      1
                ;       pad to system call vector
 0004                   rw      2*(bdos_int-1)

 0380           bdos_offset     rw      1
 0382           bdos_segment    rw      1
                        END
```

Index

Index