Copyright © 1981
Digital Research
P.O. Box 577
167 Central
Pacific Grove, CA 93950
(408) 649-3896
TWX 910 360 5001

All Rights Reserved
MP/M-86™ is a multi-user general purpose operating system. It is designed for use with any disk-based microcomputer using an Intel 8086, 8088 or compatible microprocessor with a real-time clock. MP/M-86 is modular in design, and can be modified to suit the needs of a particular installation. The hardware interface for a particular hardware environment is supported by the OEM or MP/M-86 distributor. Digital Research supports the user interface to MP/M-86 as documented in the MP/M-86 User’s Guide. Digital Research does not support any additions or modifications made to MP/M-86 by the OEM or distributor.

The MP/M-86 System Guide is intended for use by system designers who wish to modify either the user or hardware interface to MP/M-86. It therefore assumes the reader is familiar with the material covered in the Digital Research manuals that are distributed with MP/M-86. These are the MP/M-86 User's Guide, and the MP/M-86 Programmer's Guide. This document also assumes that the reader has already implemented a CP/M-86 1.0 Basic Input/Output System (BIOS), preferably on the target MP/M-86 machine.
# TABLE OF CONTENTS

1 System Overview
   1.1 MP/M-86 Organization ........................................ 2
   1.2 Memory Layout .................................................. 3
   1.3 Supervisor ..................................................... 4
   1.4 Real Time Monitor .............................................. 5
   1.5 Memory Module .................................................. 6
   1.6 Character I/O Manager ......................................... 8
   1.7 Basic Disk Operating System .................................. 9
   1.8 Extended I/O System ......................................... 10
   1.9 System Data Area ............................................. 11
   1.10 Resident System Processes ................................. 15

2 System Generation
   2.1 GENSYS Operation ............................................ 17
   2.2 System Generation Parameters ................................. 19

3 XIOS Functions
   3.1 INIT ............................................................ 25
   3.2 Entry ........................................................... 26
   3.3 Character I/O Functions ..................................... 28
   3.4 Disk I/O Function ............................................. 36
   3.5 Real-Time Monitor Functions ................................ 47
   3.7 IDLE ............................................................ 51

4 Building the XIOS
   4.1 Converting the CP/M-86 BIOS ................................ 53
   4.2 Polled Devices ................................................ 54
4.3 Interrupt Devices .......................... 55
4.4 Suggested Interrupt Handling ................. 55
  4.4.1 TICK Clock .......................... 56
  4.4.2 Uninitialized Interrupts ................. 56
4.5 Disk Definition Tables ....................... 57
  4.5.1 DPH Format .......................... 57
  4.5.2 Table Generation Using GENDEF ............. 62
  4.5.3 GENDEF Output ........................ 66
4.6 Calling MP/M-86 Functions .................... 71
4.7 Blocking/Deblocking Algorithms ............... 71
4.8 Memory Disk Application ...................... 73

5 Debugging the XIOS
  5.1 Running under CP/M-86 ....................... 75
  5.2 Running under MP/M-86 ....................... 76

6 Bootstrap and Adaptation Procedures
  6.1 The Cold Start Load Operation ............... 77
  6.2 Organization of MPM.SYS ..................... 80

APPENDICES
A BOOT ROM Listing ................................ 85
B LDBIOS Listing .................................. 95
C Example XIOS Listing ................................ 109
D Blocking and Deblocking .......................... 143
SECTION 1
SYSTEM OVERVIEW

MP/M-86 is a multi-user, real-time, general purpose Operating
System. It is designed for implementation in a large variety of
hardware environments and as such, can be easily customized to fit a
particular hardware and/or user's needs.

MP/M-86 consists of three levels of interface. They are the
user interface, the logically invariant interface, and the actual
hardware interface. The distributed form of the user interface is
the Resident System Process called the TERMINAL MESSAGE PROCESS
(TMTP). It accepts commands from the user and either initiates
transient processes, or sends messages to resident processes.

The logically invariant interface to the Operating System
consists of the system function calls as described in the MP/M-86
Programmer's Guide. This portion also interfaces transient and
resident processes with the physical interface.

The physical interface communicates directly with the
particular hardware environment. It is composed of a set of
functions that are called by a process needing physical I/O. The
relationship of the three interfaces is shown in Figure 1-1.

Digital Research distributes MP/M-86 with machine-readable
source code for both the user and hardware interfaces. The system
designer can write a new user and/or hardware interface, and quickly
incorporate them by using the system generation utility, GENSYS.

This section describes the modules that comprise a typical
MP/M-86 Operating System. It is important to understand the
material covered in this section before attempting to customize the
operating system for a particular application.
1.1 MP/M-86 Organization

The logically invariant interface of MP/M-86 is composed of four basic code modules. The Real-Time Monitor (RTM) handles process related functions including dispatching, creation and termination, as well as the logical Input/Output system. The Memory module (MEM) manages memory, and handles the memory allocation and free functions. The Character I/O module (CIO) handles all console and list device functions. The Basic Disk Operating System (BDOS) manages the file system. These four modules communicate with the Supervisor (SUP) and the Extended Input/Output System (XIOS).

The SUP module manages the user interface, as well as handling all inter-module communication. It also contains system functions that essentially call other system functions. The interface to Resident Procedure Libraries, the PROGRAM LOAD function, the COMMAND LINE INTERPRETER and the PARSE FILENAME functions are examples of the last category.

The XIOS module handles the physical interface to a particular environment. All of the logical code modules can call the XIOS to perform specific hardware dependent functions.
All code modules, including the SUP and XIOS, share a common data region called the System Data Area (SYSDAT). The beginning of the SYSDAT module has a well defined structure and is used by all code modules. Following this fixed portion are the data areas used exclusively by specific code modules. The XIOS Data Area follows all of the other code module data areas. Following the XIOS Data Area is the table area of the SYSDAT module. These tables vary in size depending on options chosen during system generation.

Resident System Processes (RSPs) are placed in memory immediately following the SYSDAT module. RSPs are selected at system generation time and are considered part of the MP/M-86 Operating System. All system data structures, like Process Descriptors (PDs), User Data Areas (UDAs) and System Queue Structures, are within their own data areas. MP/M-86 will use these structures directly if they fall within 64K of the beginning of the SYSDAT module. This guarantees space for these modules without having to consume table areas. The system manager who generates a new system does not have to be aware of the needs of RSPs that use these structures. MP/M-86 copies those system structures that fall outside of the 64K SYSDAT region into the internal system tables. This allows RSPs to occupy more area than remains in the SYSDAT region.

MP/M-86 loads all transient programs into the Transient Program Area (TPA). The TPA of a given MP/M-86 system is determined at system generation time.

1.2 Memory Layout

The MP/M-86 Operating System Area can be placed anywhere in memory except over the Interrupt Vector Area. The location of MP/M-86 is defined during system generation. The memory locations of the system modules that make up MP/M-86 are determined by GENSYS based on system generation options and the size of the modules. GENSYS places the paragraph address of each module in the System Data Area so it can be examined by using a debugger.

The XIOS Data Area must be within the System Data Area. If the XIOS is created as an 8080 Model, the code portion is combined with the data portion. The Code and Data Segments of the XIOS are set to the SYSDAT segment. If the XIOS is created as a Small Model with separate code and data, the Code Segment is placed outside of the System Data Area to allow for more table area within the 64K limit. The Data Segment always resides in the SYSDAT segment.
1.2 Memory Layout

Figure 1-2. MP/M-86 Memory Layout

1.3 Supervisor

The MP/M-86 Supervisor (SUP) manages the interaction between user programs, other system modules, the XIOS and future networking interfaces. All system function calls, whether they originate from a user program or internally from another system module, go through a common table-driven function interface. If a network module exists, it filters those functions that will be accomplished over the network. Those functions that are not intercepted are
translated into local system module functions.

The SUP module also handles system functions that use other system function calls. Functions like the PROGRAM LOAD and COMMAND LINE INTERPRETER (CUI) functions could be written as regular user programs. They accomplish no special task that a normal user program couldn't perform. These types of functions are included in the function set of MP/M-86 because the tasks are non-trivial and commonly used. Placing them in the SUP module accomplishes two goals. First, they are independent of the internal structures of the system modules, and need no modification if a different version of a system module is used. Secondly, they are outside the network interface which is designed into the SUP module. A network module need only support the more primitive functions of MP/M-86. For instance, PROGRAM LOAD would not be supported over the network as a function itself. Instead PROGRAM LOAD would make system calls for allocating memory and for reading a disk file. The network interface could intercept all access to a disk file residing in a network file system, while the memory functions would be done locally. The PROGRAM LOAD function need not be aware that its file is coming from another machine.

Table 1-1. Supervisor Functions

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>12</td>
<td>BDOS VERSION NUMBER</td>
</tr>
<tr>
<td>47</td>
<td>PROGRAM CHAIN</td>
</tr>
<tr>
<td>50</td>
<td>CALL SIGS FUNCTION</td>
</tr>
<tr>
<td>59</td>
<td>PROGRAM LOAD</td>
</tr>
<tr>
<td>167</td>
<td>RETURN SERIAL NUMBER</td>
</tr>
<tr>
<td>150</td>
<td>CALL RPL</td>
</tr>
<tr>
<td>151</td>
<td>PARSE FILENAME</td>
</tr>
<tr>
<td>152</td>
<td>GET SYSDAT ADDRESS</td>
</tr>
<tr>
<td>153</td>
<td>TIME OF DAY</td>
</tr>
<tr>
<td>163</td>
<td>MP/M VERSION NUMBER</td>
</tr>
</tbody>
</table>

1.4 Real Time Monitor

The Real-Time Monitor (RTM) handles process communications, as well as process creation, termination, and dispatching. It also handles the logical interrupt system and the device polling functions of MP/M-86. Each time a process attempts to access a resource that is not immediately available, the RTM Dispatcher takes the process out of the ready state and places it into one of the resource wait states. When another process releases the resource, the RTM Dispatcher places the waiting process back into the ready state. The STATUS byte of the Process Descriptor indicates the current state of a process. Usually, there is a System List that is associated with a given process status. When the RTM places a process on a System List, it inserts it in priority order, and after other processes with equivalent priorities. This results in priority-driven scheduling of processes on the Ready
List. Processes with equivalent priority are round-robin scheduled. At every tick of the system clock (or other interrupts that affect resources), the RTM Dispatcher takes the current process off the Ready list and reschedules it. This allows MP/M-86 to affect timeslicing.

Device Polling takes place within the RTM Dispatcher. At every dispatch, the Dispatcher polls all devices that have processes waiting. In the worst case, the Dispatcher polls each device at every system tick, although this typically happens much more often.

When a process needs to wait for an interrupt to occur, it issues a FLAG WAIT call on a logical interrupt device. When the appropriate interrupt actually occurs, the process calls the FLAG SET function which "wakes up" the waiting process. The interrupt routine then jumps to the RTM Dispatcher which reschedules the interrupted process as well as all other ready processes that have not been placed on the Ready List. At this point, the Dispatcher places the process with the highest priority into context. Typically, processes that are handling interrupts run at a high priority and therefore can react immediately.

Other functions of the Real-Time Monitor as covered in the MP/M-86 Programmer's Guide under their individual descriptions.

<table>
<thead>
<tr>
<th>Table 1-2. Real-Time Monitor Functions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function 0: SYSTEM RESET</td>
</tr>
<tr>
<td>Function 131: POLL DEVICE</td>
</tr>
<tr>
<td>Function 132: FLAG WAIT</td>
</tr>
<tr>
<td>Function 133: FLAG SET</td>
</tr>
<tr>
<td>Function 134: MAKE QUEUE</td>
</tr>
<tr>
<td>Function 135: OPEN QUEUE</td>
</tr>
<tr>
<td>Function 136: DELETE QUEUE</td>
</tr>
<tr>
<td>Function 137: READ QUEUE</td>
</tr>
<tr>
<td>Function 138: CONDITIONAL READ QUEUE</td>
</tr>
<tr>
<td>Function 139: WRITE QUEUE</td>
</tr>
<tr>
<td>Function 140: CONDITIONAL WRITE QUEUE</td>
</tr>
<tr>
<td>Function 141: DELAY</td>
</tr>
<tr>
<td>Function 142: DISPATCH</td>
</tr>
<tr>
<td>Function 143: TERMINATE</td>
</tr>
<tr>
<td>Function 144: CREATE PROCESS</td>
</tr>
<tr>
<td>Function 145: SET PRIORITY</td>
</tr>
<tr>
<td>Function 156: GET FD ADDRESS</td>
</tr>
<tr>
<td>Function 157: ABORT SPECIFIED PROCESS</td>
</tr>
</tbody>
</table>

1.5 Memory Module

The Memory Management module (MEM) handles all memory functions. MP/M-86 2.0 supports an extended fixed-partition model of memory management. Future versions of MP/M-86 may support different versions of the Memory module depending on classes of
memory management hardware that become available.

During system generation, the GENSYS program prompts the user for a list of memory partitions. The system guarantees that a single partition is never divided among unrelated programs. If any given memory request requires a memory segment that is larger than available partitions, the system joins partitions that lay next to each other to form a single contiguous area of memory. The algorithm that determines the best fit for a given allocation request takes into account the number of partitions that will be used and the amount of unused space that will be left in the memory region. This allows a system implementor or manager to decide between the tradeoffs of "internal" versus "external" memory fragmentation as described below.

"External" memory fragmentation occurs when memory is allocated in small amounts. This can lead to a situation where there is plenty of memory but there is not a contiguous area large enough to load a large program. "Internal" fragmentation occurs when memory is divided into large partitions and loading a small program leaves large amounts of unused memory in the partition. In this case a large program will always be able to load if a partition is available, but the unused areas can not be used to load a small program when all partitions are allocated.

With the MP/M-86 2.0 memory management the system implementor can specify a few large partitions, many small partitions or a combination of the two. If a particular environment requires frequently running many small programs and occasionally running large programs, memory should be divided into small partitions. This simulates dynamic memory management as the partitions become smaller. Large programs are able to load as long as memory has not become too fragmented. If the environment consists of running mostly large programs or if the programs are run serially, the large partition model should be used. The choice is not trivial and may require some experimentation before a satisfactory compromise is attained.

Memory partitions are described internally by Memory Descriptors (MDs). MP/M-86 initially places all available partitions on the Memory Free List (MFL). Once a partition (or set of contiguous partitions) is allocated, it is taken off the MFL and placed on the Memory Allocation List (MAL). The Memory Allocation List contains descriptions of contiguous areas of memory known as Memory Allocation Units (MAUs). MAUs are composed of one or more partitions. The MEM module manages the space within an MAU so that if a process requests extra memory, the module determines if the MAU has enough unused space. If it does, the extra memory requested comes from the MAU in use. A process cannot allocate memory from an MAU that it is not associated with. If a process shares memory with another process, both can allocate memory from the MAU that contains the shared memory segment. The MEM module keeps a count of how many processes own a particular memory segment to ensure that it is freed within the MAU only when no processes own it. When all of the memory within a MAU is free, the MEM module frees the MAU and
returns the memory partitions that it was composed of to the AFL. The management of unused areas within a MAU allows shared code programs to run efficiently, even when the large, fixed partition model is used.

Table 1-3. Memory Management Functions

<table>
<thead>
<tr>
<th>Function</th>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>53</td>
<td>GET MAXIMUM MEMORY</td>
<td></td>
</tr>
<tr>
<td>54</td>
<td>GET ABSOLUTE MAX MEMORY</td>
<td></td>
</tr>
<tr>
<td>55</td>
<td>ALLOCATE MEMORY</td>
<td></td>
</tr>
<tr>
<td>56</td>
<td>ALLOCATE ABSOLUTE MEMORY</td>
<td></td>
</tr>
<tr>
<td>57</td>
<td>FREE MEMORY</td>
<td></td>
</tr>
<tr>
<td>58</td>
<td>FREE ALL MEMORY</td>
<td></td>
</tr>
<tr>
<td>128</td>
<td>MEMORY ALLOCATION</td>
<td></td>
</tr>
<tr>
<td>129</td>
<td>MEMORY ALLOCATION</td>
<td></td>
</tr>
<tr>
<td>130</td>
<td>MEMORY FREE</td>
<td></td>
</tr>
</tbody>
</table>

Note: Functions 53 through 58 internally call Function 128 or 129. They are supported for compatibility with the CP/M-86 Operating System. Function 128 and 129 are equivalent.

1.6 Character I/O Manager

The Character Input/Output (CIO) module of MP/M-86 handles all console and list I/O. Every character I/O device is associated with a Character Control Block (CCB). The CCB of a device contains the current owner, the root of a linked list of PDS that are waiting for access, line editing variables and status information. CCBs reside in the CCB Table of the System Data Area. Each PD contains the CCB Index of its default console and default list device. Consoles are mapped such that CCB Index zero is associated with console zero. List Device CCBs start after the console CCBs. The number of console CCBs is taken from the XIOS MAXCONSOLE function while the number of list devices is taken from the XIOS MAXLIST function. The number of CCBs in the CCB Table is set at system generation time by GENSYS. The number of CCBs set during GENSYS must be large enough to include all list and console devices supported by the XIOS.
Table 1-4. Character I/O Functions

Function 1: CONSOLE INPUT
Function 2: CONSOLE OUTPUT
Function 3: RAW CONSOLE INPUT
Function 4: RAW CONSOLE OUTPUT
Function 5: LIST OUTPUT
Function 6: DIRECT CONSOLE I/O
Function 9: PRINT STRING
Function 10: READ CONSOLE BUFFER
Function 11: CONSOLE STATUS
Function 146: ATTACH CONSOLE
Function 147: DETACH CONSOLE
Function 148: SET DEFAULT CONSOLE
Function 149: ASSIGN CONSOLE
Function 153: GET DEFAULT CONSOLE
Function 158: ATTACH LIST
Function 159: DETACH LIST
Function 166: SET DEFAULT LIST
Function 161: CONDITIONAL ATTACH LIST
Function 162: CONDITIONAL DETACH LIST
Function 164: GET DEFAULT LIST

1.7 Basic Disk Operating System

The Basic Disk Operating System (BDOS) handles all file system functions. It is described in detail in the MP/M-86 Programmer's Guide.

Table 1-5. Basic Disk Operating System Functions

Function 13: DISK RESET
Function 14: DISK SELECT
Function 15: OPEN FILE
Function 16: CLOSE FILE
Function 17: SEARCH FOR FIRST
Function 18: SEARCH FOR NEXT
Function 19: DELETE FILE
Function 20: READ SEQUENTIAL
Function 21: WRITE SEQUENTIAL
Function 22: MAKE FILE
Function 23: RENAME FILE
Function 24: GET LOGIN VECTOR
Function 25: GET DEFAULT DISK
Function 26: SET DMA OFFSET
Function 27: GET ALLOCATION VECTOR
Function 28: WRITE PROTECT DISK
Function 29: GET READ ONLY VECTOR
Function 30: SET FILE ATTRIBUTES
Function 31: GET DISK PARAMETER BLOCK
Function 32: GET/SBT USER CODE
Function 33: RANDOM READ
Function 34: RANDOM WRITE
Function 35: GET FILE SIZE
Function 36: SET RANDOM RECORD
Function 37: RESET DRIVE
Function 38: ACCESS DRIVE
Function 39: FREE DRIVE
Function 40: WRITE RANDOM WITH ZERO FILL
Function 41: TEST AND WRITE RECORD
Function 42: LOCK RECORD
Function 43: UNLOCK RECORD
Function 44: SET MULTI-SECTOR COUNT
Function 45: SET BDDS ERROR FREE MODE
Function 46: GET DISK FREE SPACE
Function 46: FLUSH BUFFERS
Function 51: SET DMA BASE
Function 52: GET DMA
Function 100: SET DIRECTORY LABEL
Function 101: GET DIRECTORY LABEL
Function 102: READ FILE XFCB
Function 103: WRITE FILE XFCB
Function 104: SET DATE AND TIME
Function 105: GET DATE AND TIME
Function 106: SET DEFAULT PASSWORD
Function 107: RETURN SERIAL NUMBER

1.8 Extended I/O System

The Extended Input/Output System (XIOS) handles the physical interface to MP/M-86. By modifying the XIOS, MP/M-86 can be run in a large variety of physical environments. MP/M-86 recognizes two basic types of I/O devices: character devices and disk drives. Character devices are treated as serial devices that handle one character at a time, while disk drives handle random blocked I/O with a 128-byte logical sector size. Use of devices that vary from these two models must be implemented within the XIOS. In this way they appear to be standard through the XIOS interface to the other modules in MP/M-86. Sections 3 and 4 contain detailed descriptions of the XIOS functions, as well as a sample implementation.

MP/M-86 allows multiple processes to use the XIOS functions simultaneously. While the system guarantees that only one process uses a particular physical device at any given time, some XIOS functions handle more than one device and as such their interfaces must be reentrant.

An example of this is the CONSOLE INPUT function. The parameter passed to this function is the console number. There can be many processes using this function, each waiting for a character from a different console. The routines that handle the individual consoles need not be reentrant, but the common code that interfaces to these routines must be.
1.9 System Data Area

The System Data Area (SYSDAT) is the common data area for all modules of MP/M-86. The SYSDAT module is composed of three main areas. The first part is the fixed portion, containing data that is common to all modules. The second portion contains data that belongs to the individual modules. The XIOS Data Area is at the end of the second portion. The third portion of the SYSDAT module is the System Table Area which is generated and initialized at system generation by the GENSYS program.

The fixed portion of the SYSDAT module contains system-wide variables including values set by GENSYS and pointers to the individual system tables.

The format of the System Data Area is shown in Figure 1-3. The fields within the System Data Area are discussed below.
Figure 1-3. System Data Area

SUP ENTRY  Double-word address of the Supervisor entry point for intermodule communication. All internal system calls go through this entry point.

XIOS ENTRY  Double-word address of the Extended I/O System entry point for inter-module communication. All XIOS function calls go through this entry point.

XIOS INIT   Double-word address of the Extended I/O System initialization entry point. System hardware initialization takes place by a calls go through this
entry point.

**DISPATCHER** Double-word address of the Dispatcher entry point that handles interrupt returns. Executing a Far Jump to the this address is equivalent to executing an Interrupt Return instruction. The DISPATCHER routine will cause a dispatch to occur and then execute an Interrupt Return. All registers are preserved and one level of stack is used. This function should be used as a exit point by all interrupt routines that use the FLAG SET function.

**PDISP** Double-word address of the Dispatcher entry point that causes a dispatch to occur with all registers preserved. Once the dispatch is done, a RETF instruction is executed. Executing a JMPF PDISP is equivalent to executing a RETF instruction. This function should be executed whenever a resource is released that may be wanted by a waiting process.

**MPMSEG** Starting Paragraph of the Operating System Area. This is also the Code Segment of the Supervisor Module.

**RSPSEG** Paragraph Address of the First RSP in a linked list of RSP Data Segments. The first word of the Data Segment points to the next RSP in the list. Once the system has been initialized, this field is zero.

**ENDSEG** First paragraph beyond the end of the Operating System area.

**NCNS** Number of System Consoles as specified at GENSYS

**NLST** Number of List Devices as specified at GENSYS

**NCCB** Number of Character Control Blocks as specified at GENSYS

**NFLAGS** Number of System Flags as specified at GENSYS

**SYSDISK** Default System Disk. The CLI will look on this disk if it cannot open the command file on the user's current default disk.

**MMP** Maximum Memory allowed per Process

**NSLAVE** Number of Network requestors

**DAY FILE** Day File Option. If this value is OFFH, Log information is displayed on system consoles at each command. This option is chosen at GENSYS.

**TEMP DISK** Default Temporary Disk. Programs that create temporary files should use this disk. This value is specified at GENSYS.
| TICKS/SEC | The number of system ticks per second. This value is specified at GENSYS. |
| LOCKSEG | Segment Address of the BDOS Lock List |
| CCB | Address of the Character Control Block Table |
| FLAGS | Address of the Flag Table |
| MDUL | Link list root of unused Memory Descriptors |
| MPL | Link list root of Free Memory Partitions |
| PUL | Link list root of unused Process Descriptors |
| QUL | Link list root of unused Queue Descriptors |
| QMAU | Queue Buffer Memory Allocation Unit |
| RLR | Ready List Root. Linked list of PD's that are ready to run. |
| DLR | Delay List Root. Link list of PD's that are delaying for a specified number of System Ticks |
| DRL | Dispatcher Ready List. Temporary holding place for PD's that have just been made ready to run. |
| PLR | Poll List Root. Linked list of PD's that are polling on devices. |
| THRDRT | Thread List Root. Linked list of all current PD's on the system. The list is threaded through the THREAD field of the PD instead of the LINK field. |
| QLR | Queue List Root. Linked list of all System QD's. |
| MAL | Link list of Active Memory Allocation Units. A MAU is created from one or more memory partitions. |
| VERSION | Address, relative to MPMSEG of Version String |
| VERNUM | MP/M-86 Version number (Function 12) |
| MPMVERNUM | MP/M-86 Version number (Function 163) |
| TOD_DAY | Time of Day. Number of days since 1 Jan, 1978 |
| TOD_HR | Time of Day. Hour of the day |
| TOD_MIN | Time of Day. Minute of the hour |
| TOD_SEC | Time of Day. Second of the minute |
1.10 Resident System Processes

All Resident System Processes (RSPs) are considered part of the Operating System Area. At system generation, GENSYS prompts the user to select which RSPs are to be included within the Operating System. All RSPs selected are placed next to each other beginning at the end of the SYSDAT region. The advantages of an RSP are that it is permanently resident and within the Operating System Area. If the RSP creates queues or processes, the PD, QD and Queue Buffer areas are used directly by MP/M-86 instead of copying the areas into system tables. The only time these areas are copied is when the data structure is actually outside the 64K address space of the SYSDAT module. This is because, all pointers to these structures are relative to the SYSDAT segment address. Details on creating RSPs and further notes on their use are discussed in the MP/M-86 Programmer’s Guide.
SECTION 2
SYSTEM GENERATION

2.1 GENSYS Operation

MP/M-86 2.0 is generated by running the GEN SYS program under an existing CP/M-86 or MP/M-86 system. GEN SYS builds the MPM.SYS file which is an image of the MP/M-86 Operating System. MPMLDR or DDT-86 places this file in memory when debugging under CP/M-86.

GENSYS allows the user to define the hardware environment, the amount of memory to reserve for system data structures, and the selection and inclusion of Resident System Processes in the Operating System file.

A sample GEN SYS session is shown in Figure 2-1.
MP/M-86 2.0 System Generation

All Values In HEX, Defaults In Parentheses

Delete Old MPM.SYS File (N)? Y

Reading MPM Modules

Starting Paragraph of Operating System (1008) =
Number Of System Consoles (2) =
Number Of System Printers (1) =
Total Character Control Blocks (5) =
Number of Ticks Per Second (3C) =
System Drive (A) =
Temporary File Drive (A) =
Maximum Locked Records per Process (10) =
Total Locked Records in System (20) =
Maximum Open Files per Process (10) =
Total Open Files in System (20) =
Day File Logging at Console (N) =
Number Of Flags (20) =
Number Of Extra Process Descriptors (20) =
Maximum Paragraphs Per Process (FFFFFF) =
Number Of Queue Control Blocks (20) =
Size Of Queue Buffer Area in Bytes (200) =
Number Of Extra Memory Descriptors (30) =

Memory Partitions, End List With 'FFFFFF'
Starting Paragraph = 260
Length = 180
Starting Paragraph = 1800
Length = 800
Starting Paragraph = ffff

Include Resident System Processes
CLOCK (Y)?
MPMSTAT (Y)?
ECHO (Y)?
TMP (Y)?

Reading RSPs

Operating System Begins At Paragraph 1008 Ends At 19BB

**** Memory Partition Overlaps Operating System - Trimming ****
Starting Paragraph Was 1800 With Length 800
New Starting Paragraph 19BC With Length 644

** GENSYS DONE **

Figure 2-1. Sample GENSYS Session
2.2 System Generation Parameters

This section contains information relating to each of the GENSYS prompts shown in the sample session above. All the files GENSYS reads are assumed to be on the current default disk. The names of these files are shown below.

- SUP.MPM Supervisor Code Module
- RTM.MPM Real Time Monitor Code Module
- MEM.MPM Memory Manager Code Module
- CIG.MPM Character Input/Output Code Module
- BOS.MPM Basic Disk Operating System Code Module
- XIOS.MPM Extended Input/Output System Module
- SYSDAT.MPM Fixed System Data Module
- *RSP Resident System Process files. These files are optional. Those listed below are distributed with MP/M-86 2.0

- TMP.RSP TERMINAL MESSAGE PROCESS
- CLOCK.RSP CLOCK Process
- ECHO.RSP ECHO Process
- MPMSTAT.RSP System Status Process

All of the GENSYS prompts have default values except for the definitions of memory partitions. When responding with a carriage return or line feed to a question with a default value, GENSYS assumes the value in parentheses. The default values are taken from the beginning of the Data Group in the file RTM.MPM.

Delete Old MPM.SYS File (N) y

The question to delete the MPM.SYS file is always asked whether or not the file MPM.SYS exists on the current default disk. This is done to make system generation eventually possible from a submit job.

Reading MPM Modules

This response is given if the old MPM.SYS file was deleted. The new MPM.SYS file will be created using the *.MPM modules on the default disk.

Starting Paragraph of Operating System (1008) =

The starting paragraph is where the MPMLDOR is to put the Operating System. Code execution starts here, with the CS register set to this value and the IP register set to 0. The Data Segment Register is set to the beginning of the System Data Area. When first bringing up and debugging MP/M-86 under CP/M-86, the answer to this question should be 8 plus where DDT running under CP/M-86 will read (the "R" command) in a file. The following example illustrates this for the system generated by the GENSYS shown above.
The CMD Header Record created by GENSYS is at location 1000:0000 in this example. Note the ABS (absolute) field in the Code Group Descriptor (the first one) is 1008, the value given to GENSYS for the start of the Operating System. Specifically the bytes 1000:0003 and 1000:0004 which are stored low byte, high byte. The segment address of the System Data Area is the ABS field of the Data Group Descriptor, bytes 1000:12 and 1000:13. The CS and DS registers are then set to these values. The Data Segment may be verified by using the 'D' command of DDTS6, the last command shown in the above DDT session. The values displayed should correspond with the System Data Area. The XIOS segment address is at bytes 6 and 7 relative in the System Data Area, again, low byte, high byte. Break points can now be set in various XIOS routines. See the section on bringing up an XIOS for more details.

Number Of System Consoles (2) = Number Of System Printers (1) =

The number of consoles and list devices reserve a Console Control Block and a List Control Block respectively for each console and list device specified. The number of consoles is also used to compute how many TERMINAL MESSAGE PROCESS RSBS to create, one per console.

Total Character Control Blocks (5) =

The total number of CCBs reserved for physical console and list devices. This number must be greater than or equal to the maximum number of console and list devices supported in the XIOS.

Number of Ticks Per Second (3C) =

This entry value can be used by applications programs to determine the number of ticks per second. This value should reflect the number of ticks per second generated in the XIOS and may vary among MP/M-86 systems.
System Drive (A) =

The system drive where MP/M-86 looks for a transient program when it is not found on the current default drive. All the commonly used transients can be placed in one place and are not needed on every drive and user number.

Temporary File Drive (A) =

The drive entered here is used as the drive for temporary disk files. This entry can be accessed in the System Data Segment by application programs as the drive on which to create temporary files. The temporary drive should be the fastest drive in the system.

Maximum Locked Records per Process (10) =

This entry specifies the maximum number of records that a single process (usually one program) can lock at any given time. This number can range from 0 to 255 and must be less than or equal to the total locked records for the system.

Total Locked Records in System (20) =

This entry specifies the total number of locked records for all the processes executing under MP/M-86 at any given time. This number can range from 0 to 255 and should be greater than or equal to the maximum locked records per process.

It is possible to allow each process to either use up the total system lock record space, or to allow each process to lock only a fraction of the system total. The first technique implies a dynamic storage region in which one process can force other processes to block because it has consumed all available resources.

Maximum Open Files per Process (10) =

This entry specifies the maximum number of files that a single process (usually one program) can open at any given time. This number can range from 0 to 255 and must be less than or equal to the total open files for the system.

Total Open Files in System (20) =

This entry specifies the total number of open files for all the processes executing under MP/M-86 at any given time. This number can range from 0 to 255 and should be greater than or equal to the maximum open files per process.
It is possible either to allow each process to use up the total system open file space, or to allow each process to only open a fraction of the system total. The first technique implies a dynamic storage region in which one process can force other processes to block because it has consumed all available resources.

Day File Logging at Console (N) =

An affirmative response causes the generated MP/M-86 system to display the current time, file name and type, and user number of each executed command file.

Number Of Flags (20) =

Flags are mostly used for interrupt control as in MP/M-80. In MP/M-86 at least 3 flags must be specified, but more must be specified if the XDS uses interrupt-driven devices.

Number Of Extra Process Descriptors (20) =

GENSYS creates a Process Descriptor for each memory partition specified. Thus for each memory partition, at least one transient program can be loaded and run. If transient programs create "child" processes or if RSPs extend past 64K from the beginning of the System Data Area, extra Process Descriptors are needed. When first bringing up and debugging MP/M-86 the defaults for these questions will suffice.

Maximum Paragraphs Per Process (FFFF) =

A process may make MP/M-86 memory allocations. See functions 128 through 130. This question puts an upper limit on how much memory any one program can obtain. The default shown here is one mega-byte, the entire amount of memory addressable by the 8086 microprocessor.

Number Of Queue Control Blocks (20) = Size Of Queue Buffer Area in Bytes (200) =

The number of Queue Control Blocks should be the maximum number of queues that may be created by transient programs (or RSPs outside of 64K from the System Data Area). The Queue Buffer Area is space reserved for Queue Buffers. The size of the buffer area required for a particular queue is the message length times the number of messages. The Queue Buffer Area should be the anticipated maximum that transient programs will need. Again, the default values will suffice during initial system debugging.
Number of Extra Memory Descriptors (30) =

This represents the number of extra Memory Descriptors allocated for system use. GENSYS automatically generates enough MDs for normal use. More may be needed if application programs make many memory allocations, or if many shared code programs are used.

Memory Partitions, End List With "FFFF"
Starting Paragraph = 2e0
Length = 1b0
Starting Paragraph = 1000
Length = 800
Starting Paragraph = ffff

Memory partitions are highly dependent on the particular hardware environment and no defaults are given. The start and length values are paragraph values, multiply them by 16 to obtain absolute values. The memory partitions can not overlap, nor can they overlap the Operating System. GENSYS checks and trims memory partitions that overlap the Operating System but does not check for memory partitions that overlap with other memory partitions. In the example GENSYS, the second partition is trimmed and the resultant partition is displayed at the end of the GENSYS session.

Include Resident System Processes
CLOCK (Y) ?
ECHO (Y) ?
TMP (Y) ? Reading RSPs

GENSYS searches the current default disk and user for files with names ending in "RSP". There are then displayed under the 'Include Resident System Processes' header. During initial system debugging the TMP must be included. The CLOCK RSP should be included when the user is ready to debug the real-time clock.
SECTION 3

XIOS FUNCTIONS

As distributed by Digital Research, MP/M-86 2.0 is configured for operation with the Intel SBC 86/12 microcomputer, an Intel 204 diskette controller, and an Intel SBC 534 Communication Expansion Board. All hardware dependencies are concentrated in subroutines that are collectively referred to as the Extended Input/Output System, or XIOS. An MP/M-86 system implementor can modify these subroutines to tailor the system to fit almost all 8086 or 8088 disk-based operating environments. This section describes each XIOS function, and defines variables and tables referenced within the XIOS. The discussion of Disk Definition Tables is treated separately in the next section of this manual.

The explanations given in this section, assumes that the reader is familiar with the CP/M-86 BIOS. Explanations should be used in conjunction with the example XIOS listed in Appendix C.

The XIOS contains two entry points, INIT and ENTRY, at offset 0H and 3H respectively from the beginning of the XIOS code module. These entry points are described below.

3.1 INIT

The Initialization routine is called by the INIT process during system initialization. The sequence of events from the time the MPM.SYS is loaded into memory until the RSPs are created is important for understanding and debugging the XIOS Initialization.

The MPMLDR (or DDT86) loads MPM.SYS in memory at the absolute Code Segment location as indicated by the MPM.SYS file_header. The CS and DS registers are initialized to the value indicated by the Header of the MPM.SYS file (this must be done by hand if loading under CP/M-86 with DDT86). At this point, the MPMLDR jumps to location 0 and begins the Initialization code of the MP/M-86 SUP module as described below.

1) The first step of Initialization in the SUP is to setup the INIT and IDLE processes. The rest of system initialization is done under the INIT process at a priority equal to 1.

2) The INIT process calls the Initialization routine of each of the other modules with the Far Call instruction. The first instruction of each code module is assumed to be a JMP instruction to its Initialization routine. The XIOS Initialization routine is called last of these modules. Once
this call is made, the Initialization code is never used again. It may be placed in a directory buffer or other uninitialized data area.

3) The Initialization routine initializes all hardware, as shown in the example, as well as all Interrupt Vectors. Interrupt 224 is saved by the SUP module and restored upon return from the XIOS. Interrupts 1,3 and 225 are used by DDT86 and should not be initialized when debugging the XIOS with DDT86 running under CP/M-86.

4) The Initialization routine can optionally print a message to Console 0 before it executes a Far Return instruction upon completion.

5) Upon return from the XIOS, the SUP Initialization routine (running under the INIT process) creates some queues, calls the MAXCONSOLE and MAXLIST routines, and starts up the RSPs. Once this is done, the INIT process terminates.

3.2 ENTRY

All access to the XIOS (after initialization) is done through the ENTRY routine. The ENTRY routine is accessed with a Far Call to a location 3 byte from the beginning of the XIOS Code module. When the XIOS function is complete, the ENTRY routine returns with a Far Return instruction. On entry, the AL register is the function number of the routine that is to be accessed. Registers CX and DX are arguments passed to that routine. All segment registers must be maintained through the call. The example XIOS shows the DS register being placed on the stack and set to the CS register. It assumes the ES and Stack registers will be maintained by the functions being called.

<table>
<thead>
<tr>
<th>Table 3-1 XIOS Register Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Entry Parameters:</strong></td>
</tr>
<tr>
<td>AL   = function number</td>
</tr>
<tr>
<td>CX   = first parameter</td>
</tr>
<tr>
<td>DX   = second parameter</td>
</tr>
<tr>
<td>DS   = System Data Area</td>
</tr>
<tr>
<td>ES   = User Data Area</td>
</tr>
<tr>
<td><strong>Return Codes:</strong></td>
</tr>
<tr>
<td>AX   = return</td>
</tr>
<tr>
<td>BX   = AX</td>
</tr>
<tr>
<td>DS   = System Data Area</td>
</tr>
<tr>
<td>ES   = User Data Area</td>
</tr>
</tbody>
</table>
The segment registers (DS and ES) must be preserved though the ENTRY routine. When calling the SUP from within the XIOS, only the ES register must be the same. Therefore, only change the ES register locally. The following code sequence illustrates the preservation of the ES register in a block move.

```
push es
    mov es,segment_address
    rep movsw
    pop es
```

In the example XIOS, the actual functions of the XIOS are accessed through a Function Table with the function number being the actual table entry. The actual function numbers and the routines they correspond to are listed below.

<table>
<thead>
<tr>
<th>Function</th>
<th>XIOS Function Numbers</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>CONSOLE STATUS</td>
</tr>
<tr>
<td>1</td>
<td>CONSOLE INPUT</td>
</tr>
<tr>
<td>2</td>
<td>CONSOLE OUTPUT</td>
</tr>
<tr>
<td>3</td>
<td>LIST OUTPUT</td>
</tr>
<tr>
<td>4</td>
<td>PUNCH OUTPUT</td>
</tr>
<tr>
<td>5</td>
<td>READER INPUT</td>
</tr>
<tr>
<td>6</td>
<td>HOME</td>
</tr>
<tr>
<td>7</td>
<td>SELECT DISK</td>
</tr>
<tr>
<td>8</td>
<td>SET TRACK</td>
</tr>
<tr>
<td>9</td>
<td>SET SECTOR</td>
</tr>
<tr>
<td>10</td>
<td>SET DMA OFFSET</td>
</tr>
<tr>
<td>11</td>
<td>READ</td>
</tr>
<tr>
<td>12</td>
<td>WRITE</td>
</tr>
<tr>
<td>13</td>
<td>LIST STATUS</td>
</tr>
<tr>
<td>14</td>
<td>SECTOR TRANSLATE</td>
</tr>
<tr>
<td>15</td>
<td>SET DMA BASE</td>
</tr>
<tr>
<td>16</td>
<td>GET SEGMENT TABLE</td>
</tr>
<tr>
<td>17</td>
<td>POLL DEVICE</td>
</tr>
<tr>
<td>18</td>
<td>START CLOCK</td>
</tr>
<tr>
<td>19</td>
<td>STOP CLOCK</td>
</tr>
<tr>
<td>20</td>
<td>MAXIMUM CONSOLES</td>
</tr>
<tr>
<td>21</td>
<td>MAXIMUM LIST DEVICES</td>
</tr>
<tr>
<td>22</td>
<td>SELECT MEMORY</td>
</tr>
<tr>
<td>23</td>
<td>IDLE</td>
</tr>
<tr>
<td>24</td>
<td>FLUSH BUFFER</td>
</tr>
</tbody>
</table>
3.3 Character I/O Functions

********************************************
* XIOS Function 0: CONSOLE STATUS *
* Return Input Status of Specified Console *
********************************************
* Entry Parameters:
  * Register CL: Console to check
  * Returned Value:
    * Register AL: OffH if ready
    * 0 if not ready
    * BL: Same as AL
********************************************

The CONSOLE STATUS routine returns the input status of the specified console. In the example XIOS, the BX register is used to pass information to routines that call this routine internally. BX is overwritten to be the same as the AX register by the ENTRY routine on an external call.
The CONSOLE INPUT function reads a character from the specified console and returns it in register AL. The parity bit may be masked off if necessary to run application programs that expect only seven-bit ASCII. If a character is not ready, function 1 suspends the calling process until a character is typed before returning.

There is a major difference between MP/M-86 and CP/M-86 in how they wait for an event to occur. In CP/M-86, the routine typically goes into a hard loop to wait for a status to change. In MP/M-86 this is NOT recommended except during the very early stages of debugging the XIOS. There are basically two ways to wait for a hardware event to occur in MP/M-86. For non-interrupt driven devices, the POLL DEVICE method is used. For interrupt-driven devices, the FLAGSET/FLAGS WAIT method is used. These are both ways for a process to give up the CPU while waiting for an external event. This way other processes can run concurrently with I/O operations. These methods are described in the MP/M-86 Programmer’s Guide under Functions 131 through 133. The console input routines in the XIOS use the POLL DEVICE method.
The CONSOLE OUTPUT function sends the specified character to the specified console. The character is in ASCII, with the high-order parity bit set to zero.

On certain consoles, it may be necessary to ensure a delay between a carriage return line feed, or form feed and the next character. If this is the case (such as on a TI Silent 700 terminal), a variable should be set with the number of ticks that must occur before the next character can be sent. The TICK Interrupt routine can decrement this count if it is non-zero. On the next CONSOLE OUTPUT call to this console, if the count is non-zero, the DELAY function should be called with the number of ticks set to the count. Upon return, the character can be sent. Another mechanism which is more common but may induce more overhead is to use a null count. In this case, a specified number of null characters are sent to the console before the next character is sent.
The LIST OUTPUT function sends the specified character to the specified List Device. List device numbers start at 0. The LIST OUTPUT function must support the number of List devices returned by the MAILIST function. The specified character is in ASCII with zero parity.
XIOS Function 4: PUNCH OUTPUT
XIOS Function 5: READER INPUT

The Punch and Reader functions are not Supported under MP/M-86

Entry Parameters:
- Register CL: Character
- Register AL: Character

Returned Value:
- Character

The PUNCH OUTPUT and READER INPUT functions are not supported under MP/M-86. They are included in the XIOS for compatibility with CP/M-86 programs that call the DIRECT BIOS function. The PUNCH OUTPUT routine should simply return, thereby allowing programs to use the function as if the PUNCH device did not exist. The READER INPUT function should return a character 26 (\x15) which indicates the end of file.

32
XIOS Function 13: LIST STATUS

Return List Output Status

Entry Parameters:
Register CL: List Device Number

Returned Value:
Register AL: Off if Device Ready
0 if Device Not Ready
BL: Same as AL

The LIST STATUS function returns the output status of the specified list device. This function is only accessed through the CALL SIOS function.
The MAXIMUM CONSOLE function returns the number of consoles that this XIOS will support. This function may return less than the actually supported consoles but never more. The number of Character Control Blocks that will be used for consoles is determined by the return value of this routine.
XIOS Function 21: MAXIMUM LIST DEVICES

Return the Maximum number of List Devices Supported under this XIOS

Entry Parameters: None

Returned Value: Number of Consoles

The MAXIMUM LIST DEVICE function returns the number of list devices that this XIOS will support. This function may return less than the actually supported list devices but never more. The number of Character Control Blocks that will be used for list devices is determined by the value returned from this routine.
3.4 Disk I/O Functions

Disk I/O is always performed by a sequence of calls to the various disk I/O functions. These initialize the disk number to access, the track and sector on a particular disk, and the DMA offset and segment addresses involved in the I/O operation. After all these parameters are initialized, 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 the SELECT DISK function 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 it is changed. The track and sector functions are always called before the READ or WRITE operations are performed.

The READ and WRITE functions should perform several retries (10 is standard) before returning error conditions. The HOMB function may or may not actually perform the track 00 seek, depending upon the disk controller characteristics; the important point is that track 00 has been selected for the next operation, and is often treated in exactly the same manner as SETTRK with a parameter of 00.

The Disk I/O routine interfaces are the same in MP/M-86 as in CP/M-86 with the exception of the SECTRAN return register. Also, hard loops within the disk routines must be changed to either POLL DEVICE or FLAG WAITS. For initial debugging, MP/M-86 will run with the CP/M-86 BIOS disk routines with the exception of the SECTRAN register difference. Once the system runs well, all hard loops should be changed to either POLL DEVICE or FLAG WAITS. See the discussion on the CONSOLE INPUT function.

36
XIOS Function 6: HOME

Select Track 0 of the current disk

Entry Parameters: None
Returned Value: None

The HOME function returns the disk head of the currently selected disk to the track 00 position. If a disk controller does not have a special feature for finding track 00, the HOME call can be translated into a call to SETTRK with a parameter of 0.
The SELECT DISK (SELDISK) function sets the current disk drive for further operations. The Specified Disk Drive Number is 0 for drive A, 1 for drive B, and so on up to 15 for drive P. The sample XIOS supports two drives. On each disk select, SELECT DISK function must return the offset of the selected drive's Disk Parameter Header relative to the SYSDAT segment address. For standard floppy disk drives, the content of the Header and associated Tables does not change.

If there is an attempt to select a non-existent drive, SELDISK returns 0000H as an error indicator. Although SELDISK 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 SELDISK 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 that read configuration information from the disk in order to set up a dynamic Disk Definition Table.
*           XIOS Function 8: SET TRACK

* ---------------------------------------------------------------
* Set Specified Track Number

* ---------------------------------------------------------------
* Entry Parameters:
*   Register CX: Track Number
*   Returned Value: None

The SET TRACK (SETTRK) function sets the specified track number for subsequent disk accesses on the currently selected drive. The selected track may be stored in memory delaying the seek until the next READ or WRITE operation 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.
The SET SECTOR (SETSEC) function sets the specified sector number for subsequent disk accesses on the currently selected drive (see SECTRAN, below). This information may be sent to the disk controller at this point, or delayed until a READ or WRITE operation occurs.
XIOS Function 19: SET DMA OFFSET

Set Disk Memory Access Offset

Entry Parameters:
Register CX: DMA offset

Returned Value: None

The SET DMA OFFSET function sets the DMA 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 OFFH offset from the current DMA segment base, and all subsequent WRITE operations get their data from that address, until the next calls to the SET DMA OFFSET and SET DMA BASE functions occur. Note that the disk controller need not actually support direct memory access. If, for example, all data is received and sent through I/O ports, the XIOS which the user constructs will use the 1:8-byte area starting at the selected DMA offset and base for the memory buffer during the following read or write operations. Many disk controllers only support actual DMA operations to selected portions of memory. In this case, the data should be copied to the selected DMA address after the restricted physical DMA is complete.
**********************************************
* XIOS Function 11: READ
*
**********************************************
* Read a Sector from Current Drive
*
******************************************************************************
* Entry Parameters: None
* Returned Value:
* Register AL: 0 if No Error
* 1 if Physical Error
* BL: Same as AL
******************************************************************************

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 one of the following Error Codes.

  0  no errors occurred
  1  non-recoverable error condition occurred

Currently, MP/M-86 responds only to a zero or non-zero value as the Error Code. That is, if Error Code 0 then MP/M-86 assumes that the disk operation completed properly. If an error occurs, however, the XIOS should attempt several retries to see if the error is recoverable. Recovering from an error depends on the calling processes Error Mode. See the MP/M-86 system function, SBT BDOS ERROR, in the MP/M-86 Programmer's Guide for more details.
**XIOS Function 12: WRITE**

*Write a Sector to the Specified Disk*

**Entry Parameters:**
- Register CL: 0 - See Error
  1 - Codes
  2 - described
  3 - below

**Returned Value:**
- Register AL: 0 if No Error
  1 if Physical Error
  BL: same as AL

The WRITE function writes 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 and MP/M systems that use standard soft-sectord floppy drives. The Error Codes given in the WRITE command are returned in register AL, with error recovery attempts as described in the READ function. On entry to the Write function the CL register contains information to allow effective sector blocking/deblocking. The Entry Codes are listed below.

0 - deferred write
1 - Non-deferred write
2 - deferred write: 1st Sector, unallocated Block
3 - Non-deferred write: 1st Sector, unallocated Block

For additional information on the use of these Entry Codes see Section 4.7, Blocking/Deblocking Algorithms.
XIOS Function 14: SECTOR TRANSLATE

Translate Sector Number given Translate Table

Entry Parameters:
- Register CS: Logical Sector Number
- DX: Offset of Translate Table
- AX: Physical Sector Number
- BX: Same as bx

The SECTOR TRANSLATE (SECTRN) function performs logical to physical sector translation to improve the overall response of MP/M-86 systems using standard floppy drives. MP/M-86 is shipped on standard IBM 3740 8-inch floppy drives 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 the user should maintain a single density IBM-compatible version of CP/M-86 for information transfer into and out of the computer system, using a skew factor of 6.

In general, SECTRN receives a Logical Sector Number. The Logical Sector Number may range from 0 to the number of sectors -1. SECTRN also receives a Translate Table offset relative to the SYSDAT segment. The Logical Sector Number is used as a index into the Translate Table. The number found in the table is the Physical Sector Number that is to be returned. If DX = 0000H no translation takes place, and the Logical Sector Number is simply returned. Otherwise, SECTRN computes and returns the translated Sector Number. SECTRN is called even when no translation is specified in the Disk Parameter Header.
XIOS Function 15: SET DMA BASE

Set the Direct Memory Access Segment Address

Entry Parameters:
  Register CX: DMA Segment Address

Returned Value: None

The SET DMA BASE function sets the segment base for subsequent DMA read or write operations. The XIOS will use the 128-byte buffer indicated by the DMA BASE and the DMA OFFSET during READ and WRITE operations.
The FLUSH BUFFERS function indicates that all Blocking/Deblocking Buffers should be flushed. This mechanism is used whenever a process terminates, a file is closed or a disk is reset. The Error Codes given in the Flush Buffers command are returned in register AL, with recovery attempts as described in the READ function.
3.5 Real-Time Monitor Functions

Poll Specified Device and Return Status

Entry Parameters:
- Register CL: Poll Device Number

Returned Value:
- Register AL: OffH if ready
- 0 if not ready
- BL: Same as AL

The POLL DEVICE function polls a device indicated by the Poll Device Number and returns its current status. It is called at every dispatch, for each device that is being polled.

A process will poll a device only if the MP/M-86 system function 131 (POLL DEVICE) is called. The Poll Device Number used as an argument for that function is the same number that the XIOS POLL DEVICE function receives as a parameter. Typically only the XIOS will call the MP/M-86 function. The mapping of Poll Device Numbers to actual physical devices is maintained totally by the XIOS. Each polling routine must have a unique Poll Device Number. For instance, if console output and input are being polled, the console output poll routine would be associated with a different Poll Device Number than the console input poll routine.

The sample XIOS shows the POLL DEVICE function taking the Poll Device Number as an index to a table of poll functions. Once the address of the poll routine is determined, it is called and the return values are used directly for the return of the POLL DEVICE function.
The START CLOCK function enables the FLAG SET function calls on TICK interrupts. When the Operating System receives a Tick interrupt, its Interrupt Handler calls the FLAG SET function and passes it an argument of 1 (Tick Flag), while the enable condition is true. (See the use of the clockon flag in the example XIOS). The system calls START CLOCK whenever a process is delaying for a specified number of clock ticks. The system calls STOP CLOCK whenever a process is delaying for a specified number of clock ticks. The system calls STOP CLOCK when there are no processes delaying. This eliminates unnecessary processing by the TICK Process.
XIOS Function 19: STOP CLOCK

Disables Tick Flag Setting

Entry Parameters: None
Returned Value: None

The STOP CLOCK function disables the setting of the Tick Flag by the Tick interrupt routine. See Function 18.
3.6 Memory Functions

******************************************************************************
* XIOS Function 16: GET SEGMENT TABLE
*
******************************************************************************
* Not supported under MP/M-86
*______________________________________________________________
* Entry Parameters: None
* Returned Value: None

******************************************************************************
* XIOS Function 22: SELECT MEMORY
* ________________________________________________________________
* Not supported by MP/M-86 2.0
* ________________________________________________________________
* Entry Parameters: None
* Returned Value: None

The SELECT MEMORY function is not currently used by MP/M-86. In future versions of MP/M-86, this function will be used in conjunction with memory management hardware.
3.7 IDLE

*******************************
* XIOS Function 23: IDLE
* Perform Idling function
* while no processes are running
* Entry Parameters: None
* Returned Value: None
*******************************

Upon system initialization, the IDLE process will jump to the IDLE function of the XIOS. The IDLE function must never return. It must stay in a loop and use no resources that may allow it to relinquish the CPU resource. The suggested IDLE function as implemented in the sample XIOS simply calls the MP/M-86 DISPATCH function and loops. This allows polled devices to be polled by the Dispatcher if any processes are waiting for such a device. If all devices are interrupt-driven, then the loop could simply halt instead of calling the Dispatcher. In this case, the first interrupt that sets a System Flag will call the Dispatcher to allow a process to continue executing.

The IDLE routine has the lowest priority (255) available in the system. This guarantees that it will always run only when no other process is ready to run.
SECTION 4
BUILDING THE XIOS

Appendix C contains an example XIOS for MP/M-86 2.0. The XIOS is assembled and then the command file is generated as an 8080 Model program with GENCMD. In the 8080 Model XIOS, the common code and data is 'ORG'd at location 1000H. The XIOS may also be assembled and the command file generated as a Small Model program. For the Small Model XIOS, the Code Segment is 'ORG'd at 0H and the Data Segment is 'ORG'd at 1000H.

MP/M-86 accesses the XIOS through two entry points, INIT and ENTRY, at location 0 and location 3 relative to the XIOS code module (offset 1003H and 1003H for the 8080 Model, and offset 0H and 3H for the Small Model). The INIT routine is for all system hardware initialization and the ENTRY routine is for all other XIOS functions. All access to the XIOS is done through the two entry points in the XIOS with the Far Call instruction and therefore must return with a Far Return instruction. The example 8080 Model XIOS must fit within the 64K System Data Segment along with the System Data Area and Table Area. Once the source of the XIOS has been modified for a particular configuration, the following commands will generate an XIOS.MPM file for use with GRNSYS:

1. ASM86 XIOS
2. GENCMD XIOS 8080
3. REN XIOS.MPM=XIOS.CMD

4.1 Converting the CP/M-86 BIOS

The implementation of MP/M-86 for a given hardware environment assumes that a fully debugged CP/M-86 BIOS has already been implemented preferably on the target MP/M-86 machine. The implementation of CP/M-86 on the target MP/M-66 machine will also simplify debugging the XIOS using DDT86. A CP/M-86 or a running MP/M-86 system is also required for the initial generation of the MP/M-86 system when using GENSYS. The CP/M-86 BIOS may also be used as a basis for construction of the target XIOS. To transform the CP/M-86 BIOS to the MP/M-86 XIOS the following changes and additions must be made.

1. The BIOS Jump Table must be changed to use the two XIOS entry points, INIT and ENTRY. These entry points are assumed to be Jump instructions to the corresponding routines. The INIT routine takes the place of the CP/M-
86 cold start entry point and is only called during the initialization of the system following the system boot. The ENTRY routine is used as a single entry point to index into all of the XIOS functions and replaces the BIOS Jump Table. The ENTRY routine is entered with the XIOS function number in register AL. The example XIOS uses the value in the AL register as an index into a function table to obtain the address of a corresponding function.

2. A Supervisor interface must be added for execution of the MP/M-86 system functions from within the XIOS. The XIOS is considered within the Operating System and is already using the User Data Area stack. Therefore the XIOS cannot make function calls in the conventional manner. (See Section 4.6).

3. A real-time interrupt clock must be added for system resource timing, to maintain the system DELAY function and a time-of-day clock. (See Section 4.4.2).

4. The additional XIOS functions 17 through 24, listed below, must be added.

   Function 17 POLL DEVICE
   Function 18 START CLOCK
   Function 19 STOP CLOCK
   Function 20 MAXIMUM CONSOLES
   Function 21 MAXIMUM LIST DEVICES
   Function 22 SELECT MEMORY
   Function 23 IDLE
   Function 24 FLUSH BUFFER

   Each of these additional XIOS functions are described in detail in Section 3.

5. All polled devices must be changed to make use of the MP/M-86 POLL DEVICE system function. (See Sections 3.5 and 4.2, and the MP/M-86 Programmer's Guide).

6. All interrupt-driven devices must be changed to use the MP/M-86 FLAG WAIT and FLAG SET functions. (See Section 4.3 and the MP/M-86 Programmer's Guide).

4.2 Polled Devices

Polled I/O devices under the CP/M-86 BIOS will typically execute a small compute-bound instruction loop waiting for a ready status from the I/O device. If this is done in the MP/M-86 XIOS a large amount of the CPU execution time is spent in this loop. To eliminate this wasteful use of the CPU resource, the XIOS must use a system function, POLL DEVICE, to place this polling process on a Poll List. The system then polls the specified I/O device at every dispatch and returns to the polling process only when a ready status has been received. By using the POLL DEVICE function the polling process does not remain in a ready state and releases the CPU resource to other processes until a it receives a ready condition.
To do polling, a process calls the MP/M-86 POLL DEVICE function with a Poll Device Number. The system will then call the XIOS POLL DEVICE function with the same Poll Device Number at every dispatch until the device is ready. The example XIOS uses the Poll Device Number to index into a table of poll functions, calls the appropriate function and returns the I/O device status to the system.

4.3 Interrupt Devices

As is the case with handling a polled I/O device, a process handling an interrupt-driven I/O device should not execute a wait loop or a halt instruction while the process is waiting for an interrupt to occur.

Interrupt-driven devices are handled under MP/M-86 using FLAG WAIT and FLAG SET system function calls. A process that needs to wait for an interrupt to occur should make a FLAG WAIT function call with a flag number. This process will not execute until the desired Interrupt Handler makes a FLAG SET function call with the same flag number. The waiting process will then continue execution. The Interrupt Handler should follow the steps outlined below, executing a Jump Far to the Dispatcher entry point for quick interrupt response.

4.4 Suggested Interrupt Handling

Interrupt Handlers under MP/M-86 are different from those in an 8086 environment due to machine architecture differences. The example TICC Interrupt Handler should be carefully studied. During initial debugging, it is recommended that interrupts not be implemented until after the system works in a polled environment. An Interrupt Handler must perform the following basic steps:

1. Do a stack switch to a local stack. The process that was interrupted may not have enough stack space for a context save.
2. Save the register environment of the process that was interrupted.
3. Satisfy the interrupting condition. This may include resetting a hardware condition and doing a FLAGSET to notify a process that the interrupt it was waiting for has occurred.
4. Restore the register environment of the Interrupted process.
5. Switch back to the original stack.
6. If a FLAGSET function call has been made, a Jump Far to the Dispatcher entry point for Interrupt routines should be done for quicker interrupt response. This routine will call the dispatcher and then execute an IRET instruction to return from the interrupt. Otherwise if no FLAGSET call has been made, an IRET instruction should be executed to return from the interrupt.
NOTE: FLAGSET is the only system function that an interrupt routine can call.

4.4.1 TICK Clock

The XIOS must provide two time bases: a system tick for managing the Delay List and forcing dispatches, and a "one second" flag for time-of-day computation. The system TICK operation and the "one second" flag (#2) operation are logically separate even though they may physically share the same clock/timer interrupt source.

The system TICK procedure, when enabled by STARTCLOCK, must set flag #1 at system time unit intervals. The recommended time unit is a period of 16.67 milliseconds, corresponding to a tick frequency of 60 Hz. When operating with 50 Hz, use a 20 millisecond period. MP/M-86 uses the TICK procedure to manage the Delay List until the Delay List is empty, at which time the procedure is disabled by STOPCLOCK.

The system tick frequency determines the dispatch frequency for compute-bound processes. If the frequency is too high, a significant amount of system overhead is incurred by an excessive number of dispatches. If the frequency is too low, compute-bound processes will keep the CPU resource for accordingly longer periods.

The "one second" flag procedure must set Flag #2 at each second of real-time. MP/M-86 uses Flag #2 to maintain the system time and day.

4.4.2 Uninitialized Interrupts

All unused interrupts should be initialized to vector to an interrupt trap routine that prevents erroneous interrupts from vectoring to an unknown location. The example XIOS handles uninitialized interrupts by printing the Process Descriptor name that caused the interrupt followed by an uninitialized interrupt message. Then the interrupting process is unconditionally terminated.

Interrupt Vector 224 is saved prior to system initialization and restored following execution of the XIOS INIT routine. The example XIOS initializes all of the Interrupt Vectors to the uninitialized interrupt trap, then any specifically used interrupts are initialized. Interrupt 224 is left for restoration by the Operating System.

When debugging the XIOS with DDT86 running under CP/M-86, Interrupt Vectors 1, 3 and 225 should not be initialized.
4.5 Disk Definition Tables

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

4.5.1 DPH Format

In general, each disk drive has an associated (16-byte) Disk Parameter Header (DPH) 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.

```
+-------------+-------------+-------------+-------------+-------------+-------------+-------------+
| XLT | 0000 | 0000 | 0000 | DIRBUF | DPB | CSV | ALV |
+-------------+-------------+-------------+-------------+-------------+-------------+-------------+
16b 16b 16b 16b 16b 16b 16b 16b
```

Figure 4-1. Disk Parameter Header

where each element is a word (16-bit) value. The meaning of each Disk Parameter Header (DPH) element is given in Table 4-1.

<table>
<thead>
<tr>
<th>Element</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>XLT</td>
<td>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 Translation Vector.</td>
</tr>
<tr>
<td>0000</td>
<td>Scratchpad values for use within the BDOS (initial value is unimportant).</td>
</tr>
<tr>
<td>DIRBUF</td>
<td>Offset of a 128-byte scratchpad area for directory operations within BDOS. All DPHs address the same scratchpad area.</td>
</tr>
<tr>
<td>DPB</td>
<td>Offset of a Disk Parameter Block for this drive. Drives with identical disk characteristics address the same Disk Parameter Block.</td>
</tr>
<tr>
<td>CSV</td>
<td>Offset of a scratchpad area used for software check for changed disks. This offset is different for each DPH.</td>
</tr>
</tbody>
</table>

Table 4-1. Disk Parameter Header Elements
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 DPHs 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 DPH Table has the following format:

<table>
<thead>
<tr>
<th>DPHBASE</th>
<th>00</th>
<th>XLT 00</th>
<th>0000</th>
<th>0000</th>
<th>0000</th>
<th>DIRBUF</th>
<th>DBP 00</th>
<th>CSV 00</th>
<th>ALV 00</th>
</tr>
</thead>
<tbody>
<tr>
<td>01</td>
<td>XLT 01</td>
<td>0000</td>
<td>0000 + 0000</td>
<td>DIRBUF</td>
<td>DBP 01</td>
<td>CSV 01</td>
<td>ALV 01</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

(and so-forth through)

| n-1 | XLTn-1 | 0000 | 0000 | 0000 | DIRBUF | DBPn-1 | CSVn-1 | ALVn-1 |

Figure 4-2. DPH Table

where the label DPHBASE defines the offset of the DPH Table relative
to the beginning of the Operating System.

The SELDSK subroutine, defined in Section 3.4, returns 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
CMP CL,NDISK ;N BEYOND MAX DISKS?
JNB RETURN ;RETURN IF SO
0 <= N < NDISK
MOV CH,0 ;DOUBLE (N)
MOV BX,CX ;BX = N
MOV CL,4 ;READY FOR * 16
SRL BX,CL ;N = N * 16
MOV CX,OFFSET DPHBASE
ADD BX,CX ;DPHBASE + N * 16
RETURN: RET ;BX = .DPH (N)

The Translation Vectors (XLT 00 through XLTn-1) are located
elsewhere in the XIOS, and correspond one-for-one with the Logical
Sector Numbers zero through the sector count-1.

58
The Disk Parameter Block (DPB) for each drive is more complex. Each DPB, which is addressed by one or more DPHs, has the format shown in Figure 4-3.

<table>
<thead>
<tr>
<th>SPT</th>
<th>BSH</th>
<th>BLM</th>
<th>EXM</th>
<th>DSM</th>
<th>DRM</th>
<th>ALG</th>
<th>ALL</th>
<th>CKS</th>
<th>OFF</th>
</tr>
</thead>
<tbody>
<tr>
<td>16b</td>
<td>8b</td>
<td>8b</td>
<td>8b</td>
<td>16b</td>
<td>16b</td>
<td>8b</td>
<td>8b</td>
<td>16b</td>
<td>16b</td>
</tr>
</tbody>
</table>

Figure 4-3. Disk Parameter Block

where each field is a byte or word value, as shown by the "8b" or "16b" indicator below the field. The fields are defined in Table 4-2.

Table 4-2. Disk Parameter Block Fields

<table>
<thead>
<tr>
<th>Field</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>SPT</td>
<td>is the total number of sectors per track</td>
</tr>
<tr>
<td>BSH</td>
<td>is the data allocation Block Shift Factor, determined by the data block allocation size (BLS).</td>
</tr>
<tr>
<td>BLM</td>
<td>is the Block Mask which is also determined by the data block allocation size (BLS).</td>
</tr>
<tr>
<td>EXM</td>
<td>is the Extent Mask, determined by the data block allocation size and the number of disk blocks.</td>
</tr>
<tr>
<td>DSM</td>
<td>determines the total storage capacity of the disk drive.</td>
</tr>
<tr>
<td>DRM</td>
<td>determines the total number of directory entries that can be stored on this drive.</td>
</tr>
<tr>
<td>ALG, ALL</td>
<td>determine reserved directory blocks.</td>
</tr>
<tr>
<td>CKS</td>
<td>is the size of the directory checksum vector. If the high-order bit of CKS is on (i.e., &gt; 5000H), then that drive is considered to be a non-removable media and the rules for resetting that drive are modified. See the MP/M-86 Programmer's Guide.</td>
</tr>
<tr>
<td>OFF</td>
<td>is the number of reserved tracks at the beginning of the (logical) disk.</td>
</tr>
</tbody>
</table>

Although these table values are produced automatically by GENDBF, 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 block allocation size BLS, which
is not an entry in the Disk Parameter Block. The values of BSH and BLM are shown in Table 4-3 below, where all values are in decimal.

<table>
<thead>
<tr>
<th>BLS</th>
<th>BSH</th>
<th>BLM</th>
</tr>
</thead>
<tbody>
<tr>
<td>1,024</td>
<td>3</td>
<td>7</td>
</tr>
<tr>
<td>2,048</td>
<td>4</td>
<td>15</td>
</tr>
<tr>
<td>4,096</td>
<td>5</td>
<td>31</td>
</tr>
<tr>
<td>8,192</td>
<td>6</td>
<td>63</td>
</tr>
<tr>
<td>16,384</td>
<td>7</td>
<td>127</td>
</tr>
</tbody>
</table>

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>
<thead>
<tr>
<th>BLS</th>
<th>DSM &lt; 256</th>
<th>DSM &gt; 255</th>
</tr>
</thead>
<tbody>
<tr>
<td>1,024</td>
<td>0</td>
<td>N/A</td>
</tr>
<tr>
<td>2,048</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>4,096</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>8,192</td>
<td>7</td>
<td>3</td>
</tr>
<tr>
<td>16,384</td>
<td>15</td>
<td>7</td>
</tr>
</tbody>
</table>

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 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 together can be considered a string of 16-bits, as shown below.

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

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 4-5.

<table>
<thead>
<tr>
<th>BLs</th>
<th>Directory Entries</th>
</tr>
</thead>
<tbody>
<tr>
<td>1,024</td>
<td>32 times # bits</td>
</tr>
<tr>
<td>2,048</td>
<td>64 times # bits</td>
</tr>
<tr>
<td>4,096</td>
<td>128 times # bits</td>
</tr>
<tr>
<td>8,192</td>
<td>256 times # bits</td>
</tr>
<tr>
<td>16,384</td>
<td>512 times # bits</td>
</tr>
</tbody>
</table>

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 = 0FOH and ALL = 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 DPHs can address the same DPB if their drive characteristics are identical. Further, the DPB can be dynamically changed when a new drive is addressed by simply changing the pointer in the DPH since the DOS copies the DPB values to a local area whenever the SELDSK function is called.

Returning 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 XIOS. The areas must be unique for each drive, and the size of each area is determined by the values in the DPH.

The size of the area addressed by CSV is CKS bytes, which is sufficient to hold the directory checksum information for this particular drive. If CKS = (DRM+1)/4, then (DRM+1)/4 bytes must be reserved for directory checksum 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 example XIOS shown in Appendix C illustrates 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.

4.5.2 Table Generation Using GENDEF

The GENDEF utility supplied with MP/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 4-6.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Effect</th>
</tr>
</thead>
<tbody>
<tr>
<td>$C</td>
<td>Generate Disk Parameter Comments</td>
</tr>
<tr>
<td>$O</td>
<td>Generate DPBASE OFFSET $</td>
</tr>
<tr>
<td>$Z</td>
<td>280, 8080, 8085 Override</td>
</tr>
<tr>
<td>$COZ</td>
<td>(Any of the Above)</td>
</tr>
</tbody>
</table>

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. For convenience, the $O parameter produces the definition

DPBASE EQU OFFSET $

allowing a MOV CX,DPBASE in SELDSK, in order to match particular programming practices. The $Z parameter is included to override the standard 8066/8088 mode in order to generate tables acceptable for operation with 280, 8080, and 8085 assemblers.
The Disk Definition Table contained within X.DEF may be constructed with the CP/M-80 text editor, and consists of disk definition statements identical to those accepted by the DISKDEF utility supplied with CP/M-80 Version 2. A BIOS end XIOS 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 the 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 the XIOS (typically at the end), before the start of uninitialized RAM.

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

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 Physical Sector Number
skf  is the optional Sector Skew Factor
bls  is the data allocation block size
dks  is the disk size in bits 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 that 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 "bils" 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 "bils" units. That is, if the bils = 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 "bils" 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 disk reset or system reset has not occurred (when this situation is detected, MP/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 drive media is non-removable, then the value of "cks" is typically 8000H, 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 that 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 upward-compatible with CP/M-80 Version 2.0 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 423K byte disk capacity, 64 checked directory entries, and two Operating System tracks.

The DISKS statement generates a Disk Parameter Headers (DPHs), 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 DISKR 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,STRBUF,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 above. The checksum and allocation vector addresses are generated by the ENDEF statement for inclusion in the RAM area following the XIOS 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 Number from CX in the BX register. A Translation Table is constructed when the "skf" parameter is present, and the (non-zero) table address is placed into the corresponding DPHs. 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 XIOS which is loaded upon cold start, but must be available in the Operating System data segment 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)
1D80 = ENDDAT EQU OFFSET $ 013C = DAPS12 EQU OFFSET $-BEGDAT

65
which indicates that uninitialized RAM begins at offset 1C72H, ends at 1B80H-1, and occupies 013CH bytes. Note that these addresses must be free for use after the system is loaded.

After modification, the STAT program can be used to check the drive characteristics, since STAT uses the Disk Parameter Block to decode the drive information. The comment included in the LIB file by the SC parameter to GENCMD will match the output from STAT. The STAT command form

\[ \text{STAT d:DSK:} \]

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
- \( r \): Reserved Tracks

### 4.5.3 GENDEF Output

GENDEF produces a listing of the statements included in the DEF file at the user console (if 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 GENDEF are listed in Table 4-7, followed by errors that can occur when producing input and output files in Table 4-8.
Table 4-7. GENDEF Source Error Messages

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

Table 4-8. GENDEF Input and Output Error Messages

<table>
<thead>
<tr>
<th>Message</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cannot Close &quot;.LIB&quot; File</td>
<td>LIB file close operation unsuccessful, usually due to hardware write protect.</td>
</tr>
<tr>
<td>&quot;LIB&quot; Disk Full</td>
<td>No space for LIB file.</td>
</tr>
<tr>
<td>No Input File Present</td>
<td>Specified DEF file not found.</td>
</tr>
<tr>
<td>No &quot;.LIB&quot; Directory Space</td>
<td>Cannot create LIB file due to too many files on LIB disk.</td>
</tr>
<tr>
<td>Premature End-of-File</td>
<td>End of DEF file encountered unexpectedly.</td>
</tr>
</tbody>
</table>
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

gen.cmd two Sc

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.LIB

SELDSK

MOV CX,OFFSET DPBASE

INCLUDE TWO.LIB

DPBASE EQU $ ;Base of DISKS
DPE0 DW XLT0,0000H ;Transl
DW 0000H,0000H ;Scratch
DW DIRBUF,DGB0 ;Dir Bu
DW CSV0,ALV0 ;Check,
DPE1 DW XLT1,0000H ;Transl
DW 0000H,0000H ;Scratch
DW DIRBUF,DGB1 ;Dir Bu
DW CSV1,ALV1 ;Check,
DISKDEF 0,1,26,2048,2
;
;
Disk 0 in CP/M 1.4 Single Dense
4096: 128 Byte Record Capacit
512: Kilobyte Drive Capacit
128: 32 Bytes Directory Entri
256: Records / Extent
16: Records / Block
26: Sectors / Track
2: Reserved Tracks
6: Sector Skew Factor

DPB0 EQU OFFSET $ ;Disk P
DW 26 ;Sector
DB 4 ;Block
DB 15 ;Extnt
DW 255 ;Disk S
DW 127 ;Direct
DB 192 ;Alloco
DB 0 ;Alloc1
DB 32 ;Check
DW 2 ;Offset

XLT0 EQU OFFSET $ ;Transl
DB 1,7,19,19
DB 25,5,17,17
DB 23,7,9,15
DB 21,2,8,14
DB 20,26,6,12
DB 18,24,4,10
DB 16,22
IALSO EQU 32 ;Alloca
IASO EQU 32 ;Check
DISKDEF 1,1,58,2048,10
;
;
Disk 1 in CP/M 1.4 Single Dense
16384: 128-Byte Record Capacit
MP/M-86 System Guide

4.5.3 GENDEF Output

; 2048: Kilobyte Drive Capacity
; 300: 32-Byte Directory Entries
; 0: Checked Directory Entries
; 128: Records / Extent
; 16: Records / Block
; 58: Sectors / Track
; 2: Reserved Tracks

; dpbl equ offset $ ; Disk P
; dw 58 ; Sector
; db 4 ; Block
; db 15 ; Block
; db 0 ; Extnt
; dw 1023 ; Disk S
; dw 299 ; Direct
; db 248 ; Alloc0
; db 0 ; Alloc1
; dw 0 ; Check
; dw 2 ; Offset
; xltl equ 0 ; No Tra
; als1 equ 128 ; Alloc
; css1 equ 0 ; Check
; ENDEF

; Uninitialized Scratch Memory Pool
;
; begdat equ offset $ ; Start
; dirbuf rs 128 ; Direct
; alv0 rs als0 ; Alloc
; csv0 rs css0 ; Check
; alv1 rs als1 ; Alloc
; csv1 rs css1 ; Check
; enddat equ offset $ ; End of
; datsiz equ offset $-begdat ; Size of
; db 0 ; Marks

END

70
4.6 Calling MP/M-86 Functions

Routines in the XIOS cannot make system calls in the conventional manner of executing a INT 224 instruction. The conventional entry point to the RTM does a stack switch to the User Data Area (UDA) of the current process. The XIOS is considered within the Operating System and a process entering the XIOS is already using the UDA stack. Therefore, a separate entry point is used for internal system calls.

Location 3 of the SUP Code Segment is the entry point for internal system calls. Register usage for system calls though this entry point is the similar to the conventional entry point. They are as follows:

Entry:  
- CX = Function number  
- DX = Parameter  
- DS = Segment address if DX is an offset to a structure  
- ES = User Data Area  

Return:  
- AX = BX = Return  
- CX = Error Code for RTM functions  
  (BDOS functions do not use CX)  
- ES = Segment address if AX is an offset

The only differences between the internal and user entry points are the CX and ES registers on entry. CH must always be 0. ES must always point to the User Data Area of the current process. The UDA segment address can be obtained through the following code:

```
mov si,ready_list_root  
mov es,10h[si]  
```

Note: On entry to the XIOS, ES is equal to the UDA segment address.

4.7 Blocking/Deblocking Algorithms

Upon each call to the XIOS WRITE function, the MP/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 section presents a general-purpose algorithm that can be included within the XIOS 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 = deferred write sector (normal write)
1 = non-deferred write sector (directory write)
2 = deferred write to the first sector
    of a previously unallocated data block
3 = non-deferred write to the first sector
    of a previously unallocated data block

Condition 0 occurs whenever the next write operation is into a
previously written area, such as a random mode record update, when
the write is to other than the first sector of an unallocated block,
or when the write is not into the directory area.

Condition 1 occurs when a write into the directory area is
performed. Condition 2 occurs when the first record (only) of a
newly-allocated data block is written. In most cases, application
programs read or write multiple 128-byte sectors in sequence, and
thus there is little overhead involved in either operation when
blocking and deblocking records since pre-read operations can be
avoided when writing records.

Appendix D lists the blocking/deblocking algorithm in skeletal
form (the file is included on the MP/M-86 disk). Generally, the
algorithms map all MP/M-86 sector read operations onto the host disk
through an intermediate buffer that is the size of the host disk
sector. Throughout the program, values and variables that relate to
the MP/M-86 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 25 of Appendix D
define the mapping between MP/M-86 and the host system, and must
be changed if other than the sample host system is involved.

The SELDSK function clears the host buffer flag whenever a new
disk is logged-in. Note that although the SELDSK function 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 XIOS functions are READ and WRITE. These
subroutines take the place of the 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, hstrk is the host track number, and hstsec is
the host sector number (which may require translation to a physical
sector number). Note: Code must be inserted at this point that
performs the full host sector read or write into, or out of, the
buffer at hatbuf of length hatsz. All other mapping functions are
performed by the algorithms.
4.8 Memory Disk Application

The Memory Disk is a prime example of the ability of the Basic Disk Operating System to interface to a wide variety of disk drives. A 128K-byte area of memory is used to simulate a small capacity disk drive, making a very fast temporary disk. The Memory Disk is usually called the "A" drive and under GENSYS may be specified as the temporary drive, if implemented.

The example XIOS in Appendix C contains a conditional assembly switch for the necessary code needed to implement the Memory Disk. The additional Disk Parameter Blocks have been generated by adding the following command to the singles.def file used with the GENDEF utility.

    diskdef 2,1,16,,1024,127,64,0,0
SECTION 5

DEBUGGING THE XIOS

The MP/M-86 XIOS can be debugged in many different ways. This section presents two methods that require no special hardware. The first and most useful method is to load MP/M-86 as a transient program with DDT86 running under CP/M-36 or with a firmware debugger. The second method is to run DDT86 under MP/M-86. This method however, requires the majority of the system to be running and therefore is not as useful as the first.

For initial debugging, the customized XIOS should be implemented using all polled I/O devices. Also, all interrupts should be disabled including the system TICK interrupt, and Interrupt Vectors 1, 3 and 225 should not be initialized. The initial system can run without a clock interrupt and this interrupt should be implemented only after the XIOS is fully developed and tested. After the XIOS has been debugged interrupt-driven I/O devices should be implemented and tested one at a time.

Because the DDT86 debugger operates with interrupts left enabled, it is a somewhat difficult task to debug an interrupt-driven console handler. The recommended method is to leave console 0 in a polled mode while debugging the other consoles in an interrupt driven mode. Once this is done, very little, if any, debugging should be required to adapt the interrupt-driven code from another console to console 0. It is further recommended that you maintain a debugged version of your XIOS that has polled I/O for console 0. Otherwise it may not be possible to run the MP/M-86 system underneath the CP/M-86 debugger because the CP/M-36 debugger cannot get any console I/O.

5.1 Running under CP/M-86

The technique for debugging an XIOS with DDT86 running under CP/M-86 is outlined in the following steps:

1. Determine the starting paragraph where a program will be loaded when using the R command in DDT86.
2. Run GENSYS using the starting paragraph determined above plus 8 for the starting paragraph of the Operating System. This allows for the 680H-byte header in the MPM.SYS file.
3. Load the MPM.SYS file under DDT86 using the R command and setup the CS and DS registers with the BARE values found in the CMD file Header Record. See the MP/M-86 Programmer’s Guide description on the CMD file Header.
4. The addresses for the XIOS ENTRY and INIT routines can be found in the system Data Segment at offsets 28H for ENTRY

All Information Presented Here is Proprietary to Digital Research

75
and 2CH for INIT. If an 8080 Model, is used these functions will be at offset 1003H and 1004H in the System Data Segment. If a Small model is used, the Code Segment must be determined from the function addresses, and the XIOS Data Segment will be the same as the System Data Segment.

5. Begin execution of the MPM.SYS file at offset OH in the Code Segment. Break points can then be set within the XIOS for debugging.

5.2 Running under MPM/86

Debugging in this mode can be accomplished after minimal console and disk I/O handlers are running. The Code and Data Segments for the XIOS are found in the same manner as described above. It may be required to disable interrupts in this mode in order to simplify debugging.
SECTION 6

BOOTSTRAP AND ADAPTATION PROCEDURES

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

MP/M-86 is distributed on a single-density IBM-compatible 8" diskette using a file format that is compatible with all previous CP/M 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 that leads to program and data files. MP/M-86 is distributed for operation with the Intel SBC 86/12 single-board computer connected to floppy disks through an Intel 204 Controller. Three additional consoles and a list device are supported using an Intel SBC 534 communication expansion board. The operation of MP/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 (MPM.LDR)
- The MP/M-86 System (MPM.SYS)

When installed in the SBC 86/12, the BOOT ROM becomes a part of the memory address space, beginning at byte location Off000H, 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 MP/M-86. The BOOT ROM (which is the same as the CP/M-86 BOOT ROM) can be obtained from Digital Research. Alternatively, it can be programmed from the listing given in Appendix A or directly from the source file that is included on the distribution disk as BOOT.A86. The BOOT ROM reads the MPM.LDR into memory from the first two system tracks and then passes program control to the MPM.LDR for execution.

6.1 The Cold Start Load Operation

The MPM.LDR program is a simple version of CP/M-86 that contains sufficient file processing capability to read MPM.SYS from the system disk to memory. When MPM.LDR completes its operation, the MPM.SYS program receives control and proceeds to process user input commands.
Both the MPMLDR and MPM.SYS programs are preceded by the standard CMD Header Record. The 128-byte MPMLDR Header Record contains the following single Group Descriptor.

```
+-----------------------------+-----------------------------+-----------------------------+-----------------------------+
| G-Form | G-Length | A-Base | G-Min | G-Max |
+-----------------------------+-----------------------------+-----------------------------+-----------------------------+
| 1     | xxxxxxxx | 400   | xxxxxxx | xxxxxxx |
+-----------------------------+-----------------------------+-----------------------------+-----------------------------+
```

Figure 6-1. Group Descriptor - MPMLDR Header Record

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 that is offset three bytes from the beginning of the Header). Note that since only a Code Group is present, an 8080 Model is assumed. Further, although the A-Base defines the base paragraph address for MPMLDR (byte address 04000H), the MPMLDR can, in fact be loaded and executed at any paragraph boundary that does not overlap MP/M-86 or the BOOT ROM.

The MPMLDR itself consists of three parts: the Load MPM program (LDMPM), the Loader Basic Disk System (LDBIOS), and the Loader Basic I/O System (LDBIOS). Although the MPMLDR is setup to initialize MP/M-86 using the Intel 86/12 configuration, the LDBIOS can be field-altered to account for non-standard hardware using the same functions defined in the CP/M-86 BIOS. The organization of MPMLDR is shown in Figure 6-2 below:
Byte offsets from the base registers are shown at the left of the diagram. GD#1 is the Group Descriptor for the MPMLDR Code Group described above, followed immediately by a "0" group terminator. The entire MPMLDR 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 byte location 04000H where the MPMLDR program commences execution. The JMP 1200H instruction at the base of LDMMPM 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 LDMMPM program at byte offset 003H. The LDMMPM module opens the MPM.SYS file, loads the MP/M-86 system into memory and transfers control to MP/M-86 through the JMPPF MPM instruction at the end of LDMMPM execution, thus completing the cold start sequence.

The files LDMMPM.H86 and LDBDOS.H86 are included with MP/M-86 so that the user can append a modified LDBIOS in the construction of a customized loader. The example LDBIOS is listed in Appendix B for reference purposes. To construct a custom LDBIOS, modify the standard CP/M-86 BIOS to start the code at offset 1200H, and change the initialization subroutine beginning at INIT to perform disk and device initialization. Include a JMP to offset 003H at the end of the INIT subroutine. Use ASM-86 to assemble the LDBIOS.H86 program.
to produce the LDBIOS.H86 machine code file. Concatenate the three
MPMLDR modules using PIP:

\texttt{PIP MPMLDR.H86=LDMPM.H86,LDBIOS.H86,LDBIOS.H86}

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

\texttt{GENCMD MPMLDR 8080 CODE[A040]}

resulting in the file MPMLDR.CMD with a Header Record defining the
8080 Model with an absolute paragraph address of 040H, or byte
address 040H. The MPMLDR.CMD is copied to the first two tracks of a
(scratch) disk under CP/M-86 using the LDCOPY utility using the
following command:

\texttt{LDCOPY MPMLDR}

Alternately a CP/M-80 system could be used with the SYSGEN utility
using the following command.

\texttt{SYSGEN MPMLDR.CMD}

The diskette now contains an MPMLDR program that incorporates
the custom LDBIOS capable of reading the MPM.SYS file into memory.
For standardization, Digital research assumes MPMLDR executes at
byte location 0400H. MPMLDR is statically relocatable, however, and
its operating address is determined only by the value of A-Base in
the Header Record.

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

6.2 Organization of MPM.SYS

The MPM.SYS file, read by the MPMLDR program, consists of the
seven *.MPM files and included *.RSP files in CMD file format, with
a 16-byte Header Record similar to the MPMLDR program:
Figure 6-3. Group Descriptor - MPM.SYS Header Record

where, instead of a single Code Group both Code and Data Group Descriptors are used. The Code Group Descriptor has an A-Base lead address at paragraph 01008H, or byte address 10360H. The Data Group Descriptor has an A-Base immediately following the end of the code group which will vary with the modules included in that group. The entire MPM.SYS file appears on disk as shown in Figure 6-4.

Figure 6-2. MPM.SYS File Organization

where GD#1 is the Code Group Descriptor containing the A-Base value and GD#2 is the Data Group Descriptor followed by a "0" terminator.

The distributed 86/12 XIOS is listed in Appendix C, 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.DIF statements shown below:

disks 2
diskdef 0.1,20,6,1024,124,64,64,2
diskdef 1,0
endef

81
The MPM.SYS file is read by the MPMLDR program beginning at the address given by A-Base (byte address 100800H), and control is passed to the Supervisor INIT function at offset address 0000H. The actual load address of MPM.SYS is determined entirely by the addresses given in the A-Base fields which can be changed if when executing MP/M-86 in another region of memory. Note that the region occupied by the Operating System must be excluded from the memory segments defined during GENSYS. The user is recommended to regenerate the system using GENSYS to avoid any errors in the A-Base address calculations.

Similar to the MPMLDR program, the user can modify the XIOS by altering the example XIOS.A56 assembly language files that are included on the source disk. The user must create a customized XIOS that includes the specialized I/O drivers, and assemble it using ASM-86:

ASM86 BIOS

to produce the file XIOS.M86 containing the XIOS machine code. The resulting XIOS.HEX file is then converted to a CMD file (8080 Model) by executing

GENCMD XIOS 8080

If a Small Model is used, having separate code and data areas, the HEX file is converted to a CMD file by executing.

GENCMD XIOS

Finally the CMD file is renamed to an MPM file using the command

REN XIOS.MPM = XIOS.CMD

The resulting XIOS.MPM file may be place on the 8086 system disk with the other *.MPM system file ready for GENSYS.

These steps essentially complete the tailoring process. The original BOOT ROM is replaced by either the customized BOOT ROM, or a one-sector cold start loader that brings the LOADER program, with the custom LDBIOS, into memory at byte location 04000H. The MPMLDR program, in turn, reads the MPM.SYS file created by GENSYS, with the custom XIOS, into memory at the location specified in GENSYS. Control transfers to MP/M-86, and the system begins operation. MP/M-86 remains in memory until the next system reset operation takes place.

The user can avoid the two-step boot operation by constructing a non-standard disk with sufficient space to hold the entire MPM.SYS file on the system tracks. In this case, the cold start loader brings the MP/M-86 memory image into memory at the location given by A-Base, and control transfers directly to the Supervisor INIT function at offset 0000H. Thus, the intermediate MPMLDR program is eliminated entirely, although any initialization found in the LDBIOS must, of course, take place instead within the XIOS.
Since ASM-86, GENCMD and GENDEF are provided in both COM and CMD formats, either CP/M-86 or CP/M-86 can be used to aid the customizing process.
APPENDIX A

BOOT ROM Listing

******************************************************************************

* MP/M-86 uses the same BOOT ROM as CP/M-86. Therefore
* the listing is distributed with CP/M-86 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 location 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 inter-
; rupt controller is setup for inter-
; rupts 10H-17H (vectors at 000040H-00005FH)
; 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 0
; 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
; 0FP000H on the SBC 86/12 CPU board. ROM
; contains the even memory locations, and
; ROM1 contains the odd addresses. BOOT
; ROM uses RAM between 00000H and 00FFH
; (absolute) for a scratch area, along with
; the sector 1 buffer.

*****************************************************************************
OFF

000D  cr  equ  13
000A  if  equ  10

; disk ports and commands

00A0  base204  equ  0a0h
00A0  fdcom  equ  base204+0
00A0  fdcstat  equ  base204+0
00A1  fdcpum  equ  base204+1
00A1  fdcrat  equ  base204+1
00A2  fdcred  equ  base204+2
00A4  dmacadr  equ  base204+4
00A5  dmaccont  equ  base204+5
00A6  dmacscan  equ  base204+6
00A7  dmacsmdr  equ  base204+7
00A8  dmacmode  equ  base204+8
00A8  dmacstct  equ  base204+8
00A9  fdccsel  equ  base204+9
00AA  fdcsegment  equ  base204+10
00AF  reset204  equ  base204+15

; actual console baud rate

2580  baud_rate  equ  9600
0008  baud  equ  768/(baud_rate/100)

002A  csts  equ  0DAh ; 18251 status port
0028  cdata  equ  0DH ; data port

00D0  tch0  equ  020h ; 8253 PIC channel 0 port
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  lcp1  equ  0C0h ; 8259a port 0
00C2  lcp2  equ  0C2h ; 8259a port 1

; IF NOT DEBUG
ROMSEG  EQU  OFF00H ; normal
ENDIF

; IF DEBUG
; share prom with SBC 95
ROMSEG  EQU  OFF00H
ENDIF

87
This long jump prom'd in by hand

; cseg, sp, 0x3f@h, ;reset goes to here (ff)
JMPF BOTTOM ; boot is at bottom of P
EA 00 00 00 FF ; cs = bottom of prom (f
pp = 0

EVEN PROM

ODD PROM

7FB - EA 7FB - 00
7FE - 00 7FF - 00
7FA - FF ; this is not done if sp

cseg romseg

; First, move our data area into RAM at 0000:0250

0000 BCC8
0002 8ED8
0004 BESC01
0007 B80E02
000A B80D00
000D B8C0
000F B96E00
0012 F3A4
0014 B30000
0017 8ED8
0019 8ED0
001B BCA2A3
001E FC

MOV AX,0 ; data segment now in RAM
MOV DS,AX
MOV SS,AX
MOV SP,STICK_OFFSET ; initialize stack segment
CILD ; clear the direction flag

IF NOT DEBUG

; Now, initialize the console USART and baud rate

001F BOOE
0021 B6DA
0023 B60D
0025 B6DA
0027 B04B
0029 B6DA
002B B037
002D B6DA
002F BOB6
0031 B6D6
0033 B60000
0036 B6DA
0038 8AC4
003A B6D4

mov al, 0x0e ; give 8251 dummy mode
out csts, al
out csts, al
MOV 2, 46h
out csts, al
out csts, al
out csts, al
out csts, al
out csts, al
out csts, al
MOV 1, DBH
out ctmr, al
MOV ax, baud
out tch2, al
out tch2, al
out tch2, al
out tch2, al
MOV 1, 1h
out tch2, al
out tch2, al

ENDIF

; Setup the 8259 Programmable Interrupt Controller

0030 8013
MOV 1, 1h
003E E6C0 out icpl,al ;8259a ICW 1 8086 mode
        mov al,10h
0042 E6C2 out icp2,al ;8259r ICW 2 vector @ 40-5F
0044 B01F mov al,1Fh
0046 E6C2 out icp2,al ;8259a ICW 4 auto EOI master
0048 80FF mov cl,OFFh
004A E6C2 out icp2,al ;8259a OCW 1 mask all levels 0

;Reset and initialize the ISBC 204 Diskette Interface
;reset:
        out reset204,AL ;reset ISBC 204 logic end
        mov al,1
0050 E6A2 out fduration,AL ;give 8271 FDC
0052 8000 mov al,0
0054 E6A2 out fduration,AL ;reset command
0056 BB1502 mov BX,offset specs1
0059 E8100 CALL sendcom ;program
005C BB1B02 mov BX,offset specs2
005F BBB000 CALL sendcom ; Shugart SA-800 drive
0062 B21102 mov BX,offset specs3
0065 EBD500 call sendcom ; characteristics
0068 BB1022 mov BX,offset home
006B E85800 CALL execute ;home drive 0

006C BB2A03 mov bx,sector1 ;offset for first sector DMA
0071 BB0000 mov ax,0
0074 E8200 mov es,ax ;segment " " "
0076 E8700 call setup_dma

0079 BB202 mov bx,offset read0
007C E4700 call execute ;get TO S1

007F BB0603 mov es,ABS
0083 BB0000 mov bx,0 ;get loader load address
0086 EB9700 call setup_dma ;setup DMA to read loader
0089 BB0602 mov bx,offset read1
008C EB7000 call execute ;read track 0
008F BB0802 mov bx,offset read2
0092 EB3100 call execute ;read track 1
0095 BC3E802 mov leap_segment,ES
0099 C7066020006 setup far jump vector
009F FF2EE602 mov leap_offsec,0
; enter LOADER
00A9 FF2002 jmpf dword ptr leap_offset

pmsg:
00A3 8A0F mov cl,[BX]
00A5 84C9 test cl,cl
00A7 7176 jz return
00A9 EB0400 call conout

89
0043 inc BX
0045 jmp pmsg

; conout:
0080 44DA in al,csts
0082 A001 test al,1
0084 7F4F jz conout
0086 8AC1 mov al,cl
0088 E6DB out cdta,al
008A C3 ret

; conin:
00BB 44DA in al,csts
00BD A002 test al,2
00BF 7F4F jz conin
00C1 44DB in al,cdta
00C3 247F end al,7Fh
00C5 C3 ret

; execute: ;execute command string & [BX]
00C6 891E0002 mov lastcom,BX ;(BX) points to length,
00CA 687000 call sendcom ;(DAX) length<cr>
00CD 8B1C0002 mov BX,lastcom
00DE 844701 mov AL,1[BX]
00DF 243F and AL,1Fh
00E0 B90008 mov CX,0800h
00E2 32C2 cmp AL,2ch
00E4 7208 jb execpoll
00E6 B96800 mov CX,0800h
00E8 2400 and AL,0Fh
00EA 30C0 cmp AL,0Ch
00EB 7737 je return

; execpoll: ;poll for bit in b, toggled with c
00F0 E4A0 in AL,FDSTAT
00F2 22C5 and AL,CH
00F4 32C174F8 xor AL,CL "JZ execpoll"
00F6 E4A1 in AL,fdcrlst
00F8 241E and AL,1eh
00FA 7429 jz return

; zero means it was a good
00FC 3C10 cmp al,10h
MP/M-86 System Guide  Appendix A  BOOT ROM Listing

00F8 751J  ; jne fatal
00FA BB3102  ; if other than "Not Rec
00FD EE3D00  ; perform read status co
0100 E4A0  ; rd_poll:
0102 AA80  ; in al,fcc_stat
0104 75FA  ; test al,80h
0106 881E0002  ; jnz rd_poll
010A E9BDF2  ; retry
010D 4600  ; fatal:
010F 8D08  ; mov ah,0
0111 889F2702  ; mov bx,ax
0115 E888FF  ; print appropriate error message
0118 8EA0FF  ; cell pmsg
011B 56  ; cell conin
011C E92DF2  ; jmp restart
011F C7  ; then start all over
0120 B004  ; return: RET
0122 E6A8  ; return from EXECUTE
0124 B000  ; setupdma:
0126 E6A5  ; mov al,0
0128 B040  ; ; enable dma
012A E6A5  ; out dmac,AL
012C 8CC5  ; out dmac,AL
012E E6A8  ; mov AX,ES
0130 B4C4  ; out dmac,AL
0132 E6A5  ; mov AX,BX
0134 8BC3  ; mov AL,AH
0136 E6A4  ; out macadr,AL
0138 B4C4  ; mov AL,AH
013A E6A4  ; out macadr,AL
013C C3

013D E4A0  ; sendcom: routine to send a command string to $2
0141 75FA  ; in AL,fccstat
0143 8A0F  ; jnz sendcom ; insure command not busy
0145 43  ; mov CL,(BX) ; get count
0146 4A07  ; mov al,(BX) ; point to and fetch command byte
0148 EA00  ; out dccc,AL ; send command

014A FE0C  ; parmloop:
014C 74F1  ; dec CL
014E 751J  ; jz return ; see if any (more) parameters
MP/M-96 System Guide
Appendix A  Boot ROM Listing

0148 43 inc BX ;point to next parameter
parmpoll:
014F E4A0 inc AL,fdcstat
0151 2420 and AL,20h ;loop until parm not full
0153 75FA jnz parmpoll
0155 8A77 mov AL,[BX]
0157 E6A1 out fdcparm,AL ;output next parameter
0159 E9BEFF jmp parmloop ;go see about another

; Image of data to be moved to RAM
;
015C dbrombegin equ offset $015C 0000 clastcom dw 0000h ;last command

015E 03 credstring db 3 ;length
015F 52 db 52h ;read function code
0160 00 db 0 ;track 0
0161 01 db 1 ;sector 0

0162 04 credtrk0 db 4 ;
0163 53 db 53h ;read multiple
0164 00 db 0 ;track 0
0165 02 db 2 ;sectors 2
0166 19 db 25 ;through 26

0167 04 credtrkl db 4
0168 53 db 53h ;
0169 01 db 1 ;track 1
016A 01 db 1 ;sectors 1
016B 1A db 26 ;through 26

016C 026980 chome0 db 2,69h,0
016F 010C credstat0 db 1,66h
0171 033500 cspec1 db 5,35h,0dh
0174 0508F9 db 08h,08h,0e9h
0177 053510 cspec2 db 5,35h,10h
017A FFFFFF db 255,255,255
017C 053519 cspec3 db 5,35h,12h
0180 FFFFFF db 255,255,255

0183 4702 cersb1 dw offset er0
0185 4702 dw offset er1
0187 4702 dw offset er2
0189 4702 dw offset er3
018B 5702 dw offset er4
018D 6902 dw offset er5
018F 7002 dw offset er6
0191 7F02 dw offset er7
0193 9002 dw offset er8
0195 A202 dw offset er9
0197 B202 dw offset era
0199 C5C2 dw offset erb
018B D:02 dw offset e:C
019D 4702 dw offset e:R
019F 4702 dw offset e:R
01A1 4702 dw offset e:R

01A3 0DDA4E754C6C Ccr0 db cr,lf,'Null Error ?',0
20457226F72
203F3000

01A3 Ccr1 equ ccr0
01A3 Ccr2 equ ccr0
01A3 Ccr3 equ ccr0

01B8 0DDA436C6F63 Ccr4 db cr,lf,'Clock Error',0
69B20457226F72
7000

01C1 0DDA4C617465 Ccr5 db cr,lf,'Late DMA',0
204440D4100

01CC 0DDA49442047 Ccr6 db cr,lf,'ID CRC Error',0
554320457272
6F7200

01D8 0DDA444617461 Ccr7 db cr,lf,'Data CRC Error',0
204352432045
7226F7200

01EC 0DDA4E754C6C Ccr8 db cr,lf,'Drive Not Ready',0
652046E6F7420
526561647900

01FE 0DDA57726974 Ccr9 db cr,lf,'Write Protect',0
6520507267F74
65637400

0200 0DDA54726974 CcrA db cr,lf,'Trk 00 Not Found',0
30012046E6F7
20466F756864
60

0221 0DDA57726974 CcrB db cr,lf,'Write Fault',0
652046E61756C
7400

022F 0DDA53656374 CcrC db cr,lf,'Sector Not Found',0
6F72204566F74
20466F756864
50

01A3 CcrD equ ccr0
01A3 CcrE equ ccr0
01A3 CcrF equ ccr0

0242 dromend equ offset $0E6

00E6 data_length equ dromend-drombegin

; reserve space in RAM for data area
; (no hex records generated here)

0000 dseg $0E6
0200h org

0200h ram_start equ $
Appendix A  BOOT ROM Listing

02D0  lostcom   rw  1 ;last command
02D2  read0     rb  4 ;read track 0 sector 1
02D6  read1     rb  5 ;read T0 S2-26
02DB  read2     rb  5 ;read T1 S1-26
02D0  home      rb  3 ;home drive 0
02D3  rdstat     rb  2 ;read status
02D5  specs1    rb  6
02D9  specs2    rb  6
02DB  specs3    rb  6
02DE  ertrbl    rw  16
02DF  er0       rb  length cer0 ;16
02E0  er1       equ er0
02E1  er2       equ er0
02E2  er3       equ er0
02E3  er4       rb  length cer4 ;14
02E4  er5       rb  length cer5 ;11
02E5  er6       rb  length cer6 ;15
02E6  er7       rb  length cer7 ;17
02E7  er8       rb  length cer8 ;18
02E8  er9       rb  length cer9 ;16
02E9  erA       rb  length cerA ;19
02EC  erB       rb  length cerB ;14
02ED  erC       rb  length cerC ;19
02EF  er0       equ er0
02F0  er0       equ er0
02F1  er0       equ er0
02F2  er0
02F3  leap_offset rw  1
02F4  leap_segment rw  1
02F5  stack_offset equ offset $;local stack
02FA  offset $ ;local stack
02FB  sector1    equ offset $;T0 S1 read in here
02FC  offset $ ;T0 S1 read in here
02FD  Ty        rb  1 ;ABS is all we care abo
02FE  Len       rw  1
02FF  Abs       rw  1
0300  Min       rw  1
0301  Max       rw  1 ;ABS is all we care abo

94
APPENDIX B

LDDBIOS Listing

This is the LOADER BIOS, derived from the CP/M-86 BIOS program. This listing is truncated on the right, but can be reproduced by assembling the BIOS.A86 file provided with CP/M-86. Note that the distributed source file should always be referenced for the latest version.

;***********************************************************************
; Loader Basic Input/Output System (LDDBIOS)  
; for LDMPM Configured for ISBC 86/12 with  
; the ISBC 204 Floppy Disk Controller  
;  
; The only modifications of the CP/M-86  
; LDDBIOS for this MP/M-86 LDDBIOS are the  
; CCP offset and the contents of the signon  
; message, which is printed by LDMPM.A86  
; in place of the INIT routine.  
;  
; (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/MET for the 8086 or 8088 Micro-

FFFF true equ -1
0000 false equ not true

;***********************************************************************
MP/M-86 System Guide

Appendix B  LDBIOS Listing

;* Bdos_int is interrupt used for earlier *
;* versions.  *
;*
;******************************************************************************

00EO0  bdos_int   equ 224 ;reserved BDOS Interrupt

1200  bios_code  equ 1200h ;start of LDBIOS

0103  ccp_Offset  equ 0103h ;base of MPMLoader

0406  bdos_ofst   equ 0406h ;stripped BDOS entry

00DA  csts        equ 0DAh ;i8251 status port

00E8  cdata       equ 0D8h ; * data port

;******************************************************************************

;* 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+1 ;8271 in status

00A1  fdc_parm    equ base204+2 ;8271 out parameter

00A1  fdc_rslt    equ base204+3 ;8271 in result

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+9 ;8257 in status

00A9  fdc_sel     equ base204+10 ;FDC select port (n

00AA  fdc_segment equ base204+11 ;segment address re

00AF  reset_204   equ base204+15 ;reset entire inter

000A  max_retries equ 100 ;max retries on dis

000D  cr         equ 0Dh  ;carriage return

000A  lf         equ 0ah  ;line feed

org  ccpoffset

org  bios_code

;******************************************************************************

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

1200  E93C00  jmp INIT ;Enter from BOOT ROM or LOADER

96
1201 E95800 jmp WBOOT ; Arrive here from BIOS call 0
1206 E95B00 jmp CONST ; return console keyboard status
1209 E96100 jmp CONIN ; return console keyboard char
120C E96800 jmp CONOUT ; write char to console device
120F E97300 jmp LISTOUT ; write character to list device
1212 E98800 jmp PUNCH ; write character to punch device
1215 E98B00 jmp READER ; return char from reader device
1218 E99800 jmp HOME ; move to trk 0C on cur sel drive
121B E99900 jmp SKLSK ; select disk for next rd/write
121E E9C500 jmp SETTRK ; set track for next rd/write
1221 E9C700 jmp SETSEC ; set sector for next rd/write
1224 E9D000 jmp SETDMA ; set offset for user buff (DMA)
1227 E9D800 jmp READ ; read a 128 byte sector
122A E9D000 jmp WRITE ; write a 128 byte sector
122D E9E200 jmp LISTST ; return list status
1230 E9F800 jmp SECTRN ; set logical-physical sector
1233 E9F500 jmp SETMB ; set seg base for buff (DMA)
1236 E9C800 jmp GETSEG ; return offset of Mem Desc Table
1239 E94100 jmp GETIOBF ; return I/O map byte (10BYTE)
123C E94400 jmp SETIOBF ; set I/O map byte (10BYTE)

*****************************************************************************

; INITIALIZE hardware
123F 8CC8 mov ax,cs ; we entered with a JMPF so
1241 8ED0 mov ss,ax ; CS: as the initial value
1244 8ED8 mov ds,ax ; DS:
1245 8EC0 mov es,ax ; and ES:
1247 BC416 mov sp,offset stkbse ; use local stack during initialization
124A 8C07 old ; set forward direction
124B 8E push ds ; save data segment
124C 8B3000 mov ax,0 ;
124F 8E06 mov ds,ax ; point to segment zero
1251 C7080030604 mov bbos_offset,bbos_csdst ;BDOS interrupt offset
1257 ECE8B203 mov bbos_segment,CS ;BDOS interrupt segment
125B 1F pop ds ; restore data segment
125C B100 mov cl,0 ; default to dr A: on coldst
125E 8E020E jmp ccp ; jump to cold start entry o
1261 E9952E WBOOT: jmp ccp+6 ; direct entry to CCP at con

*****************************************************************************

; CP/M Character I/O Interface Routines
; Console is User task (182516) on ISBC 06/12
; at ports D8/DA

97
LDBIOS Listing

* console status
* return non-zero if RDA
 const ret:
   ret

; console input
 call const
   jr CONIN
   ; wait for RDA

; read data and remove parity
 in al, cdata
 and al, 7fh
 ret

; console output
 in al, csts
 and al, 1
 ; get console status
 ret

; wait for TBE

; Transmitter Buffer Empty
 out cdata, al
 ret

; poll list status
 ret

; not implemented in this configuration

; return EOF for now
 mov al, 1ah
 ret

; TTY: for consistency

; IOBYTE not implemented
 mov al, 0
 ret

; iobyte not implemented
 ret

; return zero in AL and flag

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

; get a console character
 call CONIN

; save and
 push ax
 mov cl, al
 call CONOUT
1296 58     pop ax     ;echo to console
1297 3c 61   cmp al,'a'
1299 72 26    jb uret     ;less than 'a' is ok
129b 3c 7a   cmp al,'z'
129d 77 02    ja uret     ;greater than 'z' is ok
129f 2c 20   sub al,'a'-'A'     ;else shift to caps
12a1 c3     ret

; utility subroutine to print messages

12a2 8a 07   mov al,[BX]     ;get next char from message
12a4 84 c0   test al,al
12a6 74 28   jz return     ;if zero return
12a8 ba cb   mov CL,al
12aa ee caff  call CONOUT     ;print it
12ad 43     inc BX
12ae ebf2   jmps pmsg     ;next character and loop

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

12b0 8b 0000 mov bx,0000h
12b2 84 f9 02 cmp cl,2     ;this BIOS only supports 2
12b6 73 18 jnb return     ;return w/ 0000 in BX if ba
12b8 b8 00 mov al, 80h
12ba 8c f9 00 cmp cl,0
12bd 75 02 jne sel1     ;drive 1 if not zero
12bf 80 40 mov al, 40h     ;else drive is 0
12c1 a2 13 15 mov sel_mask,al     ;save drive select mask
12c4 u5 60     ;now, we need disk paramete
12c6 8a d9 mov bx,cx     ;BX = word(CL)
12c8 b1 04 mov cl,4
12ca d3 e3 shl bx,cl     ;BX = word(CL)
12cc 81 c1 22 15 add bx,offset dp_base
12d0 c3     ret

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

12d1 c6 05 61 500 mov trk,0     ;move selected disk to home position (Track
12d6 bb 3b 15 mov bx,offset hom_com
12de ef 35 00 call execute
12df 74 f2 jz return     ;home drive and return if 0
12e1 ee 8b eff mov bx,offset bad_hrm     ;else print
12e4 eb e8 call pmsg     ;"Home Error"
12e6 e8 f8        jmps home     ;and retry

;******************************************************************************
; Set Track: ;Set track address given by CX
;******************************************************************************

99
MP/M-86 System Guide
Appendix B  LDBIOS Listing

12E6 880E1615  mov trk,cl ;we only use 8 bits of trac
12EA C3 ret

12EB 880E1715  mov sect,cl ;we only use 8 bits of sect
12EF C3 ret

12F0 88D9 SETSEC: ;set sector number given by cx
12F2 03DA mov bx,cx ;we only use 8 bits of sect
12F4 81F add bx,dx ;add sector to tran table a
12F6 C3 mov bl,(bx) ;get logical sector ret

12F7 890E0F15 12FB SETDRA: ;set DMA offset given by CX
12FB mov dma_adr,CX ret

12FC 890E1115 SETDAB: ;set DMA segment given by CX
1300 C3 mov dma_seg,CX ret

; GETSEGT: ;return address of physical memory table
1301 B81D15 mov bx,offset seg_table
1304 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 *
;***********************************************************************

1305 B012 READ: mov al,12h ;basic read sector command
1307 B0E2 jmps r_w_common

1309 B00A WRITE: mov al,0ah ;basic write sector command

130B B81415 r_w_common: mov bx,offset io_com ;point to command str
130E 884701 mov byte ptr l[(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

1311 891E0D15 mov last_com,BX ;save command address for r
1315 C60600C150A outer_retry: ;allow some retrying
       mov rtry_cnt,max_retries

100
retry:
    mov BX, last_com
    call send com ; transmit command to i8271
    ; check status poll

1321 8B1E0D15
1325 6A4701
1328 B50008
132B 3C2C
132D 720E
132F B90800
1332 240F
1334 3C0C
1336 B000
1338 7736
    mov BX, last_com
    mov al, [bx + 1]
    mov cx, 08000h ; mask if it will be 'int re
    cmp al, 2ch
    jb exec_poll ; ok if it is an interrupt t
    mov cx, 08000h ; else we use 'not command b
    and al, 0fh
    cmp al, 0ch
    ; unless there isn't
    mov al, 0
    ja exec_exit ; any result
    ; poll for bits in CH,
    ; toggled with bits in CL
    exec_poll:
        in al, fdcstat ; read status
        and al, ch
        xor al, cl ; isolate what we want to
        jz exec_poll ; and loop until it is done

1342 74F8
    in al, fdc_rslt ; Operation complete,
    and al, leh
    jz exec_exit ; no error, then exit
    ; some type of error occur"e

1348 7425
134A 3C10
    cmp al, 10h
    je dr_nrdy ; was it a not ready drive ?
    ; no,
    dr_rdy: ; then we just retry read or write
        dec rtry_cnt
        jnz retry
        ; up to 10 times
        ; retries do not recover from the
        ; hard error

1352 75C8
    mov ch, 0

1354 8BD8
    mov bx, ax ; make error code 16 bits
1356 B89F3B14
    mov bx, errtbl[BX]
135A E845FF
135D E4DR
    call pmag ; print appropriate message
135F ER2BFP
    in al, cdata ; flush u?art receiver buffe
    call wconnch ; read upper case console ch
1362 3C44
    cmp al, 'C'
1364 7425
1366 3C52
    je wboot 1 ; cancel
1368 74AB
    cmp al, 'R'
136A 3C49
    je outer retry ; retry 10 more times
136C 741A
    je z_ret ; ignore error
136E 0CFF
    or al, 255 ; set code for permanent err
exec_exit:
    ret

1370 C3
    ; here to wait for drive ready

101
call test_ready
jnz retry ; if it's ready row we are d

1376 281500  
call test_ready
jnz retry ; if not ready twice in row,
mov bx,offset ndymsg

1378 BBAC14  
call pmsg ; "Drive Not Ready"

137E E621FF  
ndy01:
call test_ready
jnz ndy01 ; now loop until drive ready
jmps retry ; then go retry without decre

1380 2400  
zret:
and al,0
ret ; return with no error code

1388 C3  

wboot_l: - jmp WBOOT

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

138E B640  
test sel_mask,80h

1390 F60613580  
jnz ndry2

1395 750C  
mov dh,04h ; mask for dr 0 status bit

1397 B604  
mov bx,offset rds_com

call send_com

1399 B81B5  

139C 2B0B00  
nrds2:

139F F4A0  
in al,fdc_stat ; get status word

dr_poll:

13A0 8E00  
ret ; return status of ready

13A3 75FA  

13A5 E4A1  
in al,fdc_rslt ; get "special result"

test al,dh ; look at bit for this drive

13A7 84C6  
ret

13AA E4A0  

13AC A880  
insure command not busy

13AE 75FA  

send_com:
in al,fdc_stat

13AA E4A0  
test al,80h

13AC A880  
jnz send_com ; loop until ready

13AE 75FA  

102
;see if we have to initialize for a DMA ope

mov al,[bx] ;get command byte
jne write_maybe ;if not a read it could be
jmpls init_dma ;is a read command, go set

;write_maybe:

cmp al,00h
jne dma_exit ;leave DMA alone if not read
mov cl,00h ;we have write, not read

init dma:
;we have a read or write operation, setup DMA contr

(CL contains proper direction bit)

mov al,04h
out dmac_mode,al ;enable dmac
mov al,00
out dmac_cont,al ;send first byte to con
mov al,CI
out dmac_cont,al ;load direction register
mov ax,dma adr
out dmac adr,al ;send low byte of DMA
mov ax,dma adr
out dmac adr,al ;send high byte
mov ax,dma seg
out fdc segment,al ;send low byte of segmen
mov ax,dma seg
out fdc segment,al ;then high segment addre

dma_exit:
mov cl,[bx] ;get count
inc bx
inc bx
out dmac_cont,al ;merge command and drive co
out fdc com,al ;send command byte

parm_loop:

dec cl
jz exec_exit ;no (more) parameters, retu
inc bx ;point to (next) parameter

parm_poll:
in al,fdc stat
test al,20h ;test *parameter register f
jnz parm_poll ;idle until parm reg not fu
mov al,[bx]
out fdc parm,al ;send next parameter

;jmpls parm_loop ;go see if there are more p

*********************************************************************************

;* Data Areas

*********************************************************************************

data_offset equ offset $

13FB
dseg
org data_offset ;contiguous with co
13FB DD0A000A singon db cr,lf,cr,lf
13FF 4D502FD2D238 db 'MP/M-86 Loader 2.0',cr,lf,0
36204C6F6164 657220322E30
0DDA00
1414 DD0A486F6D65 bad_hon db cr,lf,'Home Error',cr,lf,0
20457226F72 20DDA00
1423 DD0A49657465 int_trp db cr,lf,'Interrupt Trap Halt',cr,lf,0
72273707420 547261702048
616C740DDA00
143B 5B145B145B14 errtbl dw er0,er1,er2,er3
5B14
1441 6B147B148814 dw er4,er5,er6,er7
9914
144B AC14C014D214 dw er8,er9,erA,erB
B714
1453 F7145B145B14 dw erC,erD,erE,erF
5B14
145B DD0A4756C6C er0 db cr,lf,'Null Error ??','0
204572756F72
203F3F00
145B erl equ er0
145B er2 equ er0
145B er3 equ er0
146B DD0A436C6F63 er4 db cr,lf,'Clock Error :','0
6B2045727220F
72203A00
147B DD0A4C617465 er5 db cr,lf,'Late DMA :','0
204444D1202A
00
148B DD0A4442043 er6 db cr,lf,'ID CRC Error :','0
5243204572722
6F72203A00
1499 DD0A44617461 er7 db cr,lf,'Data CRC Error :','0
2043524A2045
72276F72203A
00
14AC DD0A44726976 er8 db cr,lf,'Drive Not Ready :','0
65204E6F7420
526561647920
3A00
14C0 DD0A57726974 er9 db cr,lf,'Write Protect :','0
652050726F74
65674203A00
14D2 DD0A54726B20 erA db cr,lf,'Trk 00 Not Found :','0
3030204E6F74
20466F756664
203A00
14E7 DD0A57726974 erB db cr,lf,'Write Fault :','0
104
65204661756C
74203A00
14F7 00DA53656374 erC  db cr,lf,'Sector Not Found :',0
6F72204E6F74
20466F756E64
203A00
145B erD equ er0
145B erE equ er0
145B erF equ er0
14AC ndymsg equ er8
150C 00 rtry_cnt db 0 ; disk error retry counter
150D 0000 last_com dw 0 ; address of last command string
150F 0000 dma_adr dw 0 ; dma offset stored here
1511 0000 dma_seg dw 0 ; dma segment stored here
1513 40 sel_mask db 40h ; select mask, 40h or 80h

; Various command strings for i8271
1514 03 ia_com db 3 ; length
1515 00 rd_wr db 0 ; read/write function code
1516 00 trK db 0 ; track $
1517 00 sect db 0 ; sector $
1518 022900 hom_com db 2,29h,0 ; home drive command
151B 012C rds_com db 1,2Ch ; read status command

; System Memory Segment Table
151D 01 segtable db 1 ; 1 segment
151E A901 dw tpa_seg ; seg starts after BIOS
1520 571E dw tpa_len ; and extends to 20000

; DISKS 2
1522 dpbase equ $ ; Base of Disk Param
1522 51150000 dpe0 dw xlt0,0000h ; Translate Table
1526 00000000 dw 0000h,0000h ; Scratch Area
152A 6B154215 dw dirbuf,dpb0 ; Dir Buff, Parm Blo
152E 0A16EB15 dw csvo,alv0 ; Check, Alloc Vecto
1532 51150000 dpel dw xlt1,0000h ; Translate Table
1536 00000000 dw 0000h,0000h ; Scratch Area
153A 6B154215 dw dirbuf,dpbl ; Dir Buff, Parm Blo
153E 39161A16 dw csvo,alvl ; Check, Alloc Vecto

; DISKDEF 0,1,26,6,1024,243,64,64,2
; 1944: 128 Byte Record Capacity
; 243: Kilobyte Drive Capacity
; 64: Checked Directory Entries
; 128: Records / Extent
; 8: Records / Block
; 26: Sectors / Track
; 2: Reserved Tracks
; 6: Sector Skew Factor

105
1542 dpb0 equ offset $ ; Disk Parameter Block
1542 lA60 dw 26 ; Sectors Per Track
1544 03 db 3 ; Block Shift
1545 07 db 7 ; Block Mask
1546 00 db 0 ; Extnt Mask
1547 F200 dw 242 ; Disk Size > 1
1549 3F00 dw 63 ; Directory Mask
154B C0 db 192 ; Alloc0
154C 00 db 0 ; Alloc1
154D 160 dw 16 ; Check Size
154F 0200 dw 2 ; Offset
1551 xlt0 equ offset $ ; Translate Table
1551 01070D33 db 1,7,13,19
1555 19050A1L db 25,5,11,17
1559 1703090P db 23,3,9,15
155D 1502080E db 21,2,8,14
1561 111A060C db 20,26,6,12
1565 1218040A db 18,24,4,10
156D 1016 db 16,22
001F also equ 31 ; Allocation Vector
0010 css0 equ 16 ; Check Vector Size
ordable 1,0 ;
Disk 1 is the same as Disk 0 ;
1542 dpbl equ dpb0 ; Equivalent Parameter
001F also equ also0 ; Same Allocation Vector
0010 css1 equ css0 ; Same Checksum Vector
1551 xlt1 equ xlt0 ; Same Translate Table
; ENDEF ;
; Uninitialized Scratch Memory Follows ;
156B begdat equ offset $ ; Start of Scratch Area
156B dirbuf rs 128 ; Directory Buffer
156A azl0 rs also0 ; Alloc Vector
156A cs0 rs css0 ; Check Vector
156A azl1 rs azl0 ; Alloc Vector
1569 cs1 rs css1 ; Check Vector
1569 enddat equ offset $ ; End of Scratch Area
1568 stksz equ offset $-begdat ; Size of Scratch Area
1569 00 db 0 ; Marks End of Modulus
156A loc_stk rw 32 ; Local stack for initialization
156A stkbase equ offset $ ;
156A lastoff equ offset $ ;
156A stksz equ (lastoff-0400h+15) / 16 ;
156A tps_len equ 2000h - tps_seg
156A 00 db 0 ; fill last address for GENCMD

******************************************************************************
* ;
* Dummy Data Section
* 
* 106
;****************************************************
; dseg 0 ; absolute low memory
; org 0 ; (Interrupt vectors)
0000  int0_offset  rw  1
0002  int0_segment  rw  1
  ; pad to system call vector
  rw  2*(bdos_int-1)
0380  bdos_offset  rw  1
0382  bdos_segment  rw  1
END
APPENDIX C
EXAMPLE XIOS LISTING

This is the example MP/M-86 XIOS listing. This listing has been truncated on the right, but can be reproduced by assembling the XIOS A06 file provided with CP/M-86. This BIOS allows MP/M-86 operation with the Intel 8086/16, with the BBC 284 controller and with the SBC 534 expansion interface. Use this XIOS as the basis for a customized implementation of MP/M-86.

(Notes: 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.)
**MP/M-86 System Guide Appendix C Example XIOS Listing**

---

X I O S - S 6

MP/M-86 eXtended I/O System for Intel SBC 204 Floppy Diskette Interface

Copyright (C) 1980, 1981 Digital Research, Inc.
Box 579, Pacific Grove California, 93950

The XIOS can be assembled in two forms that are acceptable to GENSYS in building an MP/M-86 II system.

8080 model:

---

Mixed code and data. The Code and Data segments are the same. The code segment is ORG’d at 1000h relative to the System Data Area.

```
high  +-----------------+
   | System Tables   |
   +-----------------+ | System Data
   | XIOS (C and D)  |
   +-----------------+ X
   | Sysdat          |
   +-----------------+
   | System Code     |
   +-----------------+ System Code

low +-----------------+
```

Separate Code and Data:

---

The Code segment is separate from the Data. The Code is ORG’d at 0000h and the Data is ORG’d at 1000h.

```
high  +-----------------+
   | System Tables   |
   +-----------------+ System Data Area
   | XIOS Data       |
   +-----------------+ X
   | Sysdat          |
   +-----------------+
   | XIOS Code       |
   +-----------------+ System Code

low +-----------------+
```

110
This XIOS is presented as an example hardware interface to an MP/M-86 system. In many places in the code, more efficient methods can be used.

(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.)

***********************************************************************

REGISTER USAGE FOR XIOS INTERFACE ROUTINES:

input:  AL = function $ (in entry)
        LX = parameter
        DX = second parameter
        DS = sysdev $ (in entry $'nc init)
        ES = CS elsewhere

output: AX = return
        BX = AX (in exit)
        ES,DS must be preserved though call

NOTE: Some changes have been made in the argument/return register usage from the CP/M-86 BIOS.

***********************************************************************

SYSTEM EQUATES

********************************************************************************

include system.def

********************************************************************************

SYSTEM DEFINITIONS

********************************************************************************

= FFFF true  equ 0ffffff ; value of TRUE
= 0000 false equ 0 ; value of FALSE
= 0000 unknown equ 0 ; value to be filled in
= 0080 dskrecl equ 128 ; l-q, disk record len
= 0620 fcblen equ 32 ; size of file control block
= 0008 pnmsiz equ 8 ; size of process name
= 0008 qnmsiz equ pnmsiz ; size of queue name
= 0008 fnmsiz equ pnmsiz ; size of file name
= 0033 ftypsiz equ 3 ; size of file type
= 00E0 mpmint equ 224 ; int vec for mpm ent.
= 00E1 debugint equ mpmint+1 ; int vec for debuggers

111
ulen     equ  0100h  ; size of $data
pdlen    equ  030h  ; size of $Process Descriptor
todlen   equ  5     ; size of $Time of Day struct
flag_tick equ 1     ; flag 0 = tick flag
flag_sec equ 2     ; flag 1 = second flag
flag_min equ 3     ; flag 2 = minute flag
ldtabsiz equ 0aah  ; ldtablelen=11, 10 entries

; conditional assembly switches

debug    equ false
****MDISK SUPPORT****
FFFF     equ true
***************

000D     cr   equ 0dh  ; carriage return
000A     lf   equ 0ah  ; line feed

;***********************************************************************
;
; CHARACTER I/O EQUATES
;
;***********************************************************************
;
; base address of serial board
0040     serbase equ 040h
;
0004     nconsoles equ 4
0001     nlists  equ 1

;
; using Intel 8255 serial board
;
004F     serreset equ serbase+0fh
;
004E     setestmode equ serbase+0eh
;
004C     ctcenable equ serbase+0ch
;
004D     uartenable equ serbase+0dh
;
;counter/timer mode addresses
0043     ctc0to2md equ serbase+03h
0047     ctc3to5md equ serbase+07h
0036     ctc0mode  equ 036h
0076     ctc1mode  equ 076h
0086     ctc2mode  equ 066h
0036     ctc3mode  equ 036h
counter/timer load addresses
; and count values
0040  ctc0Old  equ  serbase+00h
0041  ctc1Old  equ  serbase+01h
0042  ctc2Old  equ  serbase+02h
0044  ctc3Old  equ  serbase+04h
0008  cnt0Valh  equ  008h
0000  cnt0Valh  equ  000h
0008  cnt1Valh  equ  008h
0000  cnt2Valh  equ  000h
0008  cnt2Valh  equ  000h
0000  cnt3Valh  equ  000h
0000  cnt3Valh  equ  000h

; uart mode and command
0040  w0mode  equ  04eh
0040  w1mode  equ  04eh
0040  u0mode  equ  04eh
0040  u1mode  equ  04eh
0037  u0cmd  equ  037h
0037  u1cmd  equ  037h
0037  u2cmd  equ  037h
0037  u3cmd  equ  037h

; console i/o and status ports
; in and out status masks

0059  c0ioprt  equ  008h
00DA  c0stpport  equ  00ah
0002  c0inmsk  equ  02h
0002  c0outmsk  equ  01h

0042  c1ioprt  equ  serbase+02h
0042  c1stpport  equ  serbase+03h
0002  c1inmsk  equ  002h
0001  c1outmsk  equ  001h

0044  c2ioprt  equ  serbase+04h
0045  c2stpport  equ  serbase+05h
0002  c2inmsk  equ  002h
0001  c2outmsk  equ  001h

0046  c3ioprt  equ  serbase+06h
0047  c3stpport  equ  serbase+07h
c31msk  equ       002h
0001  c3outask  equ       001h
       ;   list 0
       ;
0040  l0ioprt   equ       serbase+00h
0041  l0stport  equ       serbase+01h
0002  l01msk    equ       002h
0001  l0oumsk   equ       001h
       ;
;******************************************************************************************
;*    DISK I/O EQUATES
;*
;*    Intel iSBC 204 Disk Controller Ports
;*
;******************************************************************************************
00A0  base204   equ       ja0h
       ;SBC204 assigned addr
00A6  fdcom     equ       base204+0 ;271 FDC out command
00A0  fdstat    equ       base204+0 ;271 in status
00A1  fdparm    equ       base204+1 ;271 out parameter
00A1  fdrst     equ       base204+1 ;271 in result
00A2  fdrst     equ       base204+2 ;271 out reset
00A4  dmacadr   equ       base204+4 ;2575 DMA base addr out
00A5  dmaccon   equ       base204+5 ;2575 out control
00A6  dmacscn   equ       base204+6 ;2575 out scan control
00A7  dmacadr   equ       base204+7 ;2575 out scan address
00A8  dmacmode  equ       base204+8 ;2575 out mode
00A8  dmacstat  equ       base204+9 ;2575 in status
00A9  fcsel     equ       base204+9 ;FDC select port
00AA  fdccm     equ       base204+10 ;segment addr register
00AF  resetc04  equ       base204+15 ;reset interface
500A  max_retries equ       10 ;retries on disk i/o
       ;before per error
2000  mdiskbase  equ       2000h
       ;base address of mdisk
;******************************************************************************************
;*    SUP/RTM EQUATIONS
;*
;******************************************************************************************
0100  tracebit   equ       0100h
0088  f_dispatch equ       142 ;MMP dispatch func #
008F  f_terminate equ       143 ;MMP terminate func #
0083  f_polldev  equ       133 ;MMP polldev func #
0085  f_flagset  equ       133 ;MMP flagset func #
p_flag equ word ptr 06h ; PD flag field
p_name equ byte ptr 08h ; PD Name field
p_cns equ byte ptr 020h ; PD console field
pf_keep equ 02h ; KEEP bit in p_flag

; flag assignments

tick_flag equ 1
sec_Flag equ 2

device #$ assignments for POLL DEV

c0indev equ 00h ; console 0 input device
c0outdev equ 04h ; console 0 output device
c0indev equ 02h ; console 2 input device
c0outdev equ 05h ; console 1 output device
c2indev equ 02h ; console 2 input device
c2outdev equ 06h ; console 2 output device
c3indev equ 03h ; console 3 input device
c3outdev equ 07h ; console 3 output device
fiphy_poll_dev equ 09h ; floppy disk poll device

; system data area must precede code
; area for 8080 model of the XIOS

; include sysdat.lib

;******************************************************************************
; System Data Area
;******************************************************************************

org 00h supmod rw 2
org 038h dispatcher equ (offset $)
org 040h mpmseg rw 1
org 044h endseg rw 1
org 068h rlr rw 1 ; Ready List Root
org 072h thrdrt rw 1 ; Process Thread Root
org 074h qlr rw 1 ; Queue List Root

115
MP/M-86 System Guide

Appendix C Example XIOS Listing

= 0078
  version  rw  1 ; MP/M-86, version in SUP
= 007A
  vernum  rw  1 ; MP/M-86 w/BDOS V3.0
= 007C
  mpmvernum  rw  1 ; MP/M-86 Version 1.0
= 007E
cod  rb  5 ; Time of day Structure

  org 01000h

1000 endsysdat  equ ((offset $)+0fh) AND 0fff0h

CSEG
  org offset endsysdat

;***********************************************************************
; ; SYSTEM CODE AREA
; ; XIOS JUMP TABLE
; ;***********************************************************************

1000 E9DA00
1003 E9E101
  jmp init ; system initialization

  JMP entry ; xios entry point

1006 0000
1008

  sysdat  dw  0 ; Sysdat Segment

1008

  supervisor  equ offset $

  rw  2

;***********************************************************************
; ; UTILITY SUBROUTINES
; ;***********************************************************************

;===
;===

; pmsg:
; print message on current console until null (char 0)
; input: BX = address of message
; ; put running processes console
; ; number in DL

100C 9CFA

100C FF362E18

1012 C7B62E18FFFF

1018 182688180610

101E 288B368D00

1023 8A5420

1026 1F

ploop:

1027 8A07

1029 3C00740C

102D 8AC8

  pushf

  pushf

  pushf

  push ds

  mov stoppoll, true

  push ds

  mov si,dlr

  mov si,dl

  push ds

  mov al,[bx]

  mov al,0

  cmp al,0

  je pmsg_ret

  mov cl,al

  ; CL = character
102F 5253  push dx | push bx ; save console, posit.
1031 EBFC01  call conout  ; print it
1034 5B5A  pop bx | pop dx ; restore posit., cons.
1036 43EBE2  inc bx | jmps ploop ; inc and loop

pmmsg_ret:
1039 8F062E18  pop stoppoll
103D 9DC3  popf | ret ; end of message

;****************************
;*                        
;*  INTERRUPT Routines    
;*                        
;****************************

;these variables must be in code segment

103F 0000  tickInt_ss  dw  0
1041 0000  tickInt_sp  dw  0
1043 0060  ax_save    dw  0
1045 0060  zero       dw  0

;=============
tickint:
;=============
; interrupt handler for tick interrupts

1047 1E2E8E120610  push ds | mov ds, sysdat
104D 28B8C163F10  mov tickInt_ss, ss
1052 2E89264110  mov tickInt_sp, sp
1057 2EA34310  mov ax_save, ax
105B 8C08  mov ax, cs
105D 8600  mov ss, ax
105F BC6715  mov sp, offset tickInt_ias
1062 2EFP364310  push ax_save
1067 535152  push bx | push cx | push dx
106A 5565706  push bp | push si | push di | push es
106E 8ED8  mov ds, ax

; check to set second flag

1070 F80331B75D  dec tick_count | jnz do_tick_flag
1076 C50631B83C  mov tick_count, 60
1079 BA0020  mov dx, sec_flag
107E B185E87601  mov cl, f_flagset | call supif
do_tick_flag:
; check to set tick flag

1083 B03E3018FF75  cmp clockon, true | jne tick_done
1088 BA0100  mov dx, tick_flag
108D B185E86701  mov cl, f_flagset | call supif
tick_done: ; restore context

117
1092 075F5E5D  pop es ! pop di ! pop si ! pop bp                
1096 05A95B58  pop dx ! pop cx ! pop bx ! pop ax                
109A 2E8E163F10 mov ss, tickint_ss                       
109F 2E8B264110 mov sp, tickint_sp
10A4 1F       force dispatch                          
            ; jmp intdisp
            intdisp:                                        
=================================================================================
10A5 2EFF2E3800 jmpf cs:dword ptr .dispatcher
=================================================================================
            ; unknown interrupts go here ...
=================================================================================
            ; We will terminate the process that caused this
            ; after writing a message to the process's default
            ; console. If the process is in KEEP mode, we will
            ; force it to terminate anyway ...
            ; We don't need to save any registers since we are
            ; not going to return to the process.

10AA 08CB    mov ax, cs                                   
10AC 2E8E1B0610 mov ds, sysdat                          

10B1 2E8B1E680C mov bx, rir                              
10B5 83C308  add bx, p_name                             
10B9 C647063A mov byte ptr 6[bx], ','                 
10BD C647070C mov byte ptr 7[bx], 0                  
10C1 E84BFF  call pmsg                                   

10C4 B80C15  mov bx, offset int_trp                      
10C7 E842FF  call pmsg                                   

10CA 2E8B1E6800 mov bx, rir                              
10CF 81E706FDFF and p_flag[bx], not pf_keep             
10D4 B9F000  mov cx, f_termiate                         
10D7 FAFFFF  mov dx, 0xffff                             
10DA CDE0   int 224                                     
10DC P4     hit ; hard stop                               
            ; the terminate returned !!!!!

**************************************************************************************

INITIALIZATION CODE AREA

Inter-Module Interface Routines

118
**MP/M-86 System Guide**

**Appendix C Example XIOS listing**

```assembly
;******************************************************************************
;***
;init:     XIOS system initialization routine.
;***
The INIT routine initializes all necessary hardware.
;***
; -called from SUP init. routine with CALLP
; -Interrupt 224 is reinitialized by SUP later
; -It is okay to turn on interrupts at any time
; a STI is performed immediately after RETP
; -Current Stack has about 10 levels here. Must do a
;   local stack switch if more is needed.
; -If assembled (GENCMD'd) with 8080 model,
;   CS=DS=Syssdat
; -If assembled with separate Code and Data,
;   CS=Code (ORGed at 0) DS=Syssdat
; -This example shows 8080 model
; input:   DS = syssdat segment address
;******************************************************************************
;-----------------------------------------------------------------------------
;SYSTEM INITIALIZATION AREA
;-----------------------------------------------------------------------------
; ;Save syssdat seg addr in case we need ;to see system data area. Set DS=CS
10DD 1E  push ds  ;save DS on stack for ;exit
10DE 000018  ;initialize segment registers
  mov syssdat,ds  ; save syssdat for ; syssdat access
  ;place copy of SUPMOD in data segment
  ;into Code Segment (supervisor)
10E3 BB0000  mov bx,offset supmod
10E6 B1B100  mov si, supervisor
10E9 8A072E898400  mov ax,(bx) ; mov cs:[si],ax
10F0 BB47022E8984 0200  mov ax,2[bx] ; mov cs:2[si],ax
 ;Make copy of Interrupt Routines
 ;access point to dispatcher in ;Code Segment
10F8 5E1F  push cs  ; pop ds  ; DS = CS
```
10FA FC  cld  ;set forward direction
  
  10FB 8C166715  mov initias,ss
  10FF 89266915  mov initasp,sp
  ;103 8C8B8ED0  mov ax,cs  mov ss,ax
  1107 BCAB15  mov sp,offset initstack
  
  ; Setup all interrupt vectors in low
  ; memory to address trap
  if not debug
  110A 1E  push ds  ;DS must be saved
  110B 06  push es
  110C B80000  mov ax,0
  110F B8DB  mov ds,ax
  1111 B8CD  mov es,ax  ;set ES and DS to zero
  ;setup interrupt 0 to trap routine
  1113 C7060000AA10  mov .6,offset int_trap
  1119 B0B0200  mov .2,CS
  111D AF4000  mov di,4
  1120 B0C000  mov si,0  ;then propagate
  1123 B9FE01  mov cx,510  ;trap vector co
  1126 F3A5  rep movsw  ;int 253
  1128 07  pop ez
  1129 1E  pop ds  ;restore DS,ES
  endif

;------------------------------------------------------------------
; CHARACTER I/O Initialization
;
;*************************************************************************
;  *  sbc 534 serial board initialization
;*************************************************************************

112A B600  mov al,0
112C B64F  out werset,al  ;reset serial board
112E E64E  out setestmode,al  ;set test mode to off
1130 E64C  out ctcenable,al  ;enable ctc addressing

129
; start clock for port 0
1132 B036E643 mov al,ctc0mode | out ctc0t02md,al
1136 B008E640 mov al,ctc0vall | out ctc0old,al
113A B000E640 mov al,ctc0vsh | out ctc0old,al
;
; start clock for port 1
113E B076E643 mov al,ctc1mode | out ctc1t02md,al
1142 B008E641 mov al,ctc1vall | out ctc1ld,al
1146 B000E641 mov al,ctc1vsh | out ctc1ld,al
;
; start clock for port 2
114A B0B6E643 mov al,ctc2mode | out ctc2t02md,al
114E B008E642 mov al,ctc2vall | out ctc2ld,al
1152 B000E642 mov al,ctc2vsh | out ctc2ld,al
;
; start clock for port 3
1156 B036E647 mov al,ctc3mode | out ctc3t05md,al
115A B008E644 mov al,ctc3vall | out ctc3ld,al
115E B000E644 mov al,ctc3vsh | out ctc3ld,al
;
1162 E64D out uartenable,al ; enable UART addressing
;
; initialize port 0
1164 B04EE641 mov al,u0mode | out 10stport,al
1168 B037E641 mov al,u0cmd | out 10stport,al
;
; initialize port 1
116C B04EE641 mov al,u1mode | out c1stport,al
1170 B037E643 mov al,u1cmd | out c1stport,al
;
; initialize port 2
1174 B04EE645 mov al,u2mode | out c2stport,al
1178 B037E645 mov al,u2cmd | out c2stport,al
;
; initialize port 3
117C B04EE647 mov al,u3mode | out c3stport,al
1180 B037E647 mov al,u3cmd | out c3stport,al
;
;********************************************************************************
; DISK I/O INITIALIZATION
;********************************************************************************

;***** MDISK SUPPORT *****
if memdisk
;initialize MDISK

1184 B90020 mov cx,mdiskbase
1187 06BE61 push es | mov es,cx
118A BF00088E5E5 mov dl,0 | mov ax,0e5e5h
1190 2639057405 cmp es:[dl],ax | je mdisk_end
1195 B90020 mov cx,2000h
1198 F3AB rep stos ax

119A 07 pop es

121
END

; SUP/RTM INITIALIZATION

; Initialize Clock Tick

if not debug

; set up tick interrupt vector
	tick causes interrupt 22

	sub ax, ax
	push ds : mov ds, ax
	nov word ptr .080h, offset tick_int

	mov word ptr .08ah, cs
	pop ds

	; secup ticks to occur every
	1/60th of a second

	mov tick_count, 6c

	mov al, 03h ! out 0d6h, al ; init ch 0 = mode 2

	; set # of 1/1.2288e6 seconds

	; 20480 = 5000h = 1/60th second

	mov al, 000h ! out 0d0h, al ; low count

	mov al, 005h ! out 0d0h, al ; high count

	; set up interrupt controller

	mov al, 013h ! out 0c0h, al ; ICW1

	mov al, 020h ! out 0c2h, al ; ICW2

	; = base interrupt

	mov al, 00fh ! out 0c2h, al ; ICW4,

	; auto EO1,

	; 8086 mode

	mov al, 00fh ! out 0c2h, al ; OCW2,

	; interrupt mask,

	; only 1

endif

; ------------------------------------------

; INITIALIZATION EXIT

; ------------------------------------------

; allow poll_device mechanism to work


; mov scoppoll, false

; print optional message on Console 0


; restore stack

; all stack switches must be in

; critical areas (interrupts off).


; mov bx, offset signon

call pmsg


; pushf ! pop ax

; cli

; mov ss, initss

; mov sp, initsp

122
push ax
popf

;return back to BDOS

pop ds
retf

;******************************************************************************
; *
; *       ENTRY POINT CODE AREA
; *
;******************************************************************************

entry:       ; XIOS Entry Point
;====           ===========
; All calls to the XIOS routines enter through here
; with a CALLF. Must return with a RETF
; input:  AL = function number
;        CX = parameter
;        DX = 2nd parameter
; output: AX = BX = return

clcd         ;clear D flag
mov bx,cs    ;mov ds,bx
            ;(only 8080 model)
mov ah,0     ;shl ax,1
            ;call routine
mov bx,ax    ;call functab[bx]
mov bx,ax    ;BX=AX
retf         ;All Done

input:       ;CX = function #
;        DX = parameter
;        DS = parameter segment if address
;        ES = user data area
; output: BX = AX = return
;        CX = error code for RTM functions
;        ES = return segment if address

mov ch,0
callf cs:dword ptr .supervisor ! ret

;******************************************************************************
; *
; *       MP/M XIOS functions
; *
;******************************************************************************

;******************************************************************************
; *
; *       CHARACTER I/O CODE AREA
; *
;******************************************************************************
;======
const:   ; Function 0: Console Status
;======
; input: CL = console device number
; output: AL = 0ffh if ready
;      = 000h if not ready
1201 B50D1E1  mov ch,0 ! shl cx,1
1205 B8D9   mov bx,cx
1207 BB97AC15 mov dx, const tbl[bx]
120B BADE   mov bl,dh ;BL = status mask
120D B600   mov dh,0 ;DX = status port address

; find input status for console device
120F EC     in al,dx
1210 22C3   and al,bl
1212 B000   mov al,0
1214 7402   jz badstatus
1216 B0FF   mov al,0ffh
badstatus:  ret

;======
conin:    ; Function 1: Console Input
;======
; input: CL = console device number
; output: AL = character
1219 B50D1E1  mov ch,0 ! shl cx,1
121D B8D9   mov bx,cx
121F BB97BC15 mov dx, con int tbl[bx]
1223 BADE   mov bl,dh ;BL = poll device no.
1225 B500   mov dh,0 ;DX = i/o port address

; input routine for console device
1227 B70052  mov bh,0 ! push dx
122A E81C925A call rt m_poll ! pop dx
122E EC     in al,dx
          ;and al,07fh ;CP/NET uses parity bit
122F C3    ret

;======
conout:   ; Function 2: Console Output
;======
; input: CL = character
; output: None
1230 B60D1E2  mov dh,0 ! shl dx,1
1234 BBDA   mov bx,dx
1236 BB97C415 mov dx, con out tbl[bx]

124
123A 8ADE  mov bh,0 ;BL = poll device no.
123C B600  mov dh,0 ;DX = i/o port address

; output routine for console device
123E 5152  push cx ! push dx
1240 B7068B0402 mov bh,0 ! call rtm_poll
1245 5A58  pop dx ! pop ax
1247 EEC3  out dx,al ! ret

;=====
plist:  ; Function 3: List Output
;======
; input:  CL = character
;        DL = console device 
; output: None
1249 B600DIE2 mov dh,0 ! shl dx,1
124D 8BD8  mov bx,dx
124F 8897CE15 mov dx,lowutbl[bx]
1253 8ADE  mov bl,dh ;BL = poll device no.
1255 B600  mov dh,0 ;DX = list port address

; output routine for list device
1757 5152  push cx ! push dx
1259 B7068B0402 mov bh,0 ! call rtm_poll
125E 5A58  pop dx ! pop ax
1260 EEC3  out dx,al ! ret

;=====
punch:  ; Function 4: Punch Output
reader:  ; Function 5: Reader Output
;======
PUNCH and READER devices are not supported
; input:  CL = character
; output: AL = character (control Z)
1262 B01AC3 mov al,lah ! ret ; return EOF

;=====
listst:  ; Function 13: List Status
;======
; input:  CL = list device number
; output: AL = 0ffh if ready
;        = 000h if not ready
1265 3500D1E1 mov ch,6 ! shl cx,1
1269 8BD9  mov bx,cx
126D 8897CC15 mov dl,lowutbl[bx]
1270F 8ADE  mov bl,dh ;BL = status mask
1271 B600  mov dh,0 ;DX = output port address

; find output status of List device

125
in al, dx
and al, bl
cmp al, bl
mov al, 0ffh
jx gstat
mov al, 0

;=======
maxconsole; ; Function 20: Maximum Consoles
;=======
; input: None
; output: AL = number of consoles
mov ax, nconsoles
ret

;=======
maxlist; ; Function 21: Maximum List Devices
;=======
; input: None
; output: AL = number of consoles
mov ax, nlists
ret

;******************************************************************************************
;*
; DISK I/O CODE AREA
;*
;******************************************************************************************

;===
; Home
;===
; move selected disk to home position (Track 0)
; if there is hardware home function, it should be done here otherwise, do a settrk to 0
; input: None
; output: None
mov trk, 0
set disk i/o to track zero
xor bx, bx
mov bl, disk
;index into disk home routine
shl bx, 1
jmp dskhomtbl[bx]

flpy_home:
mov bx, offset hom_com
call execute ;Home drive

hom_cet:
ret ; and return

;=======
SELDISK; ; Function 7: Select Disk
;=======

126
; input: CL = disk to be selected
; output: AX = offset of DPH relative from
; XIOS Data Segment

129F 33C0 xor ax,ax ;zero registers ax and bx
12A1 8BD8 mov bx,ax
12A3 B500 mov ch,0 ;translate logical disk
12A5 08F1 mov si, cx ;to physical disk device
12A7 8A34B516 mov al,dtrnttbl[si]
12AB 3CFF cmp al,0fh ;valid disk select?
12AD 740C jz sel_ret ;if not valid return

12AF A28416 mov disk,al ;save physical drive no.
12B2 B104 mov cl,4
12B4 D2E0 shl al,cl ;multiply by 16
12B6 05D716 add ax, offset dp_base
12B9 8BD8 mov bx,ax

sel_ret:
12BB B8CJ mov ax,bx
12BD C3 ret

;=====
; SETTRK: ; Function 8: Set Track
;=====
; input: CX = Track Number
; output: None

12BE 880EAB16 mov trk, cl ;we only use 8 bits of
12C2 C3 ; track address
ret

;=====
; SETSEC: ; Function 9: Set Sector
;=====
; input: CX = Sector Number
; output: None

12C3 880EAC16 mov sect, cl ;we only use 8 bits of
12C7 C3 ; sector address
ret

;=====
; SETDMA: ; Function 10: Set DMA Offset
;=====
; input: CX = Offset of DMA buffer
; output: None

12C8 890EA416 mov dma_addr, cx
12CC C3 ret

;******************************
;* All disk I/O parameters are setup: the *

127
; read and write entry points transfer one
; sector of 128 bytes to/from the current
; DMA address using the current disk drive
;*******************************************************************************

;===
READ:    ; function 11: read
;===
input: None
output: AL = 00h if no error occurred
       = 01h if error occurred

12CD 33DB    xor bx,bx
12CF 8A1BB416 mov bl,disk ;index into disk read routine
12D0 8E E3    shl bx,1
12D5 FFA7CB16 jmp diskrtbl[bx]

;***** MDISK SUPPORT *****
if memdisk

; mdisk_read:
12D9 E8E00    call mdisk_calc
12DC 883EA416E00 mov di,dma_addr ! mov si,0
          00
12E7 069B06A616 push es ! mov es,dma_seg
12E8 18B8D8 push ds ! mov ds,ax
12EB 944000F3A5 mov cx,64 ! rep movs ax,ax
12F0 1F07 pop ds ! pop es
12F2 860000C3 mov ax,0 ! ret

; endif
;***************************

; flpy_read:
12F6 A012    mov al,i2h ; basic read sector command
12F8 EB46    jmp$ r_w_common

;***** MDISK SUPPORT *****
if memdisk

; mdisk_calc:
12FA B7008A1EAB16 mov bh,0 ! mov bl,trak
1300 881A00F7E3 mov ax,26 ! mul bx
1305 B7008A1EAC16 mov bh,0 ! mov bl,sect
130B 03C3B103 add ax,bx ! mov cl,3
130F D3E050120 shl ax,cl ! add ax,mdiskbase+1
1314 C3 ret

; endif
;********************

;===
WRITE:    ; function 12: write
;===

;***** MDISK SUPPORT *****
if memdisk

; mdisk_write:
;***** MDISK SUPPORT *****
if memdisk

; endif
;********************

;===

; input:  CL = 0 - deferred write
;       1 - non-deferred write
;       2 - def wrt lst sect unalloc blk
;       3 - non-def wrt lst sect unalloc blk
; output: AL = 00h if no error occurred
;       01h if error occurred

1315 33DB       xor bx,bx
1317 8AEE416     mov bl,disk   ;index into disk write routine
131B D1E3       shi bx,1
131D FF7D116     jmp dskwrtnbl[bx]

;----- MDISK SUPPORT -----
if memdisk

1321 E8D6FF     call mdisk_calc
1324 BF00008B36A4 mov dl,0 ! mov si,dma_adr
                 16
1328 086C0       push es! mov es,ax
132E 1E8E1EA616  push ds! mov ds,dma_seg
1333 B9400039A5  mov cx,64 ! rep movs ax,ax
1338 1F07       mov ds! pop es
133A 880000C3    mov ax,0 ! ret
endif

;**************************************************************

133E B00A       mov al,0ah   ;basic write sector command
                r_w_common:

1340 B8A916     mov bx,offset io com ;point to command string
1343 884701     mov byte ptr 1[BX],al ;put command into string

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

1346 B080       mov al,00h
134E F606B41601 test disk,1    ;A drive is even phys. drive
1352 7502       jnz exec1    ;B drive is odd phys. drive
1354 B040       mov al,40h
1355 A2A816     exce:  mov sel_mask,al
1357 891EA216   mov last_com,BX ;save command address
                ;for retries
                outer_retry: ;allow some retrying
                    mov rtry_cnt,max_retries
                retry:       mov BX,last_com
1361 E86700     call send_com  ;transmit command to i8271

129
; check status poll

1364 8B1EA216
1368 8A4701
136B 890008
136E 3C2C
1370 7208
1372 89B080
1375 240F
1377 3C0C
1379 B000
137B 7731

flpy_poll:

mov bx,flpy_poll_dev
mov bx,flpy_poll_dev
; get command op code
mov al,[bx]
mov cx,0800h
; mask if it will be "int ret"
cmp al,2ch
jbl flpy_poll
; ok if it is an interrupt to
mov cx,0800h
; else use "not command bus"
and al,0fh

137D 8900B216
1381 8B009E008C200
1387 E4A1
1389 241E
138B 7421
138D 3C10
138F 7508
1391 B1B00
1394 7503
1396 B1600
1399 FE9EA116
139D 75BE

; toggled with bits in CL
; polled for bits in CH,

cmp al,10h

139F 241E
13A1 6400
13A3 88D8
13A5 BBFD015
13A9 EB6F0C
13AC 0CFF
13AE C3

; some type of error occurred

jnz exec_exit

13AF B640
13B1 FF606A81680

; poll for bits in CH,

ja exec_exit

; any result

; toggled with bits in CL

; operation complete,

in al,fdc_rslt

; see if result indicates
and al,leh

; an error

jz exec_exit

; no error, then exit

; some type of error occurred

cmp al,10h

1399 FPGA116

; was it a not ready drive ?
;jnz dr_rdy

; yes, here to wait for drive ready

; call test_ready

call test_ready

; if not ready twice in row,

jnz dr_rdy

; if ready try again

call test_ready

dr_rdy:
; then we just retry read or write

dec rtrty_cnt

; retrys do not recover from the

jnz rtrty

; hard error

; up to 10 times

; setup error table index

and al,leh

; make error code 16 bits

mov bx,ax

; print appropriate message

mov bx,errno[BX]
call pmmsg

; set code for permanent error

or al,255

exec_exit:

ret

;******************************

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

;******************************

test_ready:

mov dh,40h

; proper mask if dr 1

mov al,255

; test sel_mask,80h

130
jnzd nrdy2
mov dh, 04h ;mask for dr 0 status bit

nrdy2:
mov bx,offset rds_com
call send_com

dr_poll:
in al,fdc_stat ;get status word
test al,80h
jnzd dr_poll ;wait for not command busy

in al,fdc_rslt ;get "special result"
test al,dh ;look at bit for this drive
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:
in al,fdc_stat
test al,80h ;insure command not busy
jnzd send_com ;loop until ready

;check to initialize for a DMA operation

mov al,1,(bx) ;get command byte
cmp al,12h
jne write_maybe ;if not read it maybe write

mov cl,40h ;is a read command, go set DMA
jmps init_dma

write_maybe:
cmp al,0ah
jne dma_exit ;leave DMA if not read/write
mov cl,80h ;we have write, not read

init_dma:
;read or write operation, setup DMA controller
 
; (CL contains proper direction bit)

mov al,04h
out dmac_mode,al ;enable dmac
out dmac_cont,al ;1st byte to ctrl port
mov al,cl
out dmac_cont,al ;load direction register
mov ex,dmaadr
out dmacadr,al ;send low byte of DMA
mov al,ah
out dmacadr,al ;send high byte
mov ox,dma_seg
out fdc_segment,al ;send low byte of seg addr
mov al,ah
out fdc_segment,al ;then high segment address
MP/M-86 System Guide  Appendix C  Example XIOS Listing

dma_exit:
  1400 8AOF  mov  cl, [BX] ; get count
  1402 43   inc  BX
  1403 9A07  mov  al, [BX] ; get command
  1405 9A06A816 or  al, sel_mask ; merge command and drive code
  1409 E6A0  out  fdc_com, al ; send command byte

parm_loop:
  140B FEC9  dec  ci
  140D 749F  jnz  exec_exit ; no (more) parameters, return
  140F 43   inc  BX ; point to (next) parameter

parm_poll:
  1410 E4A0  in  al, fdc_stat
  1412 A820  test  al, 20h ; "parameter register full"
  1414 75FA  jnz  parm_poll ; idle until parm reg not fu
  1416 8A07  mov  al, [BX]
  1418 E6A1  out  fdc_parm, al ; send next parameter
  141A E8EF  jmps parm_loop ; see if more parameters

flipy_poll_stat:
;~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
; See if current operation complete

141C 2EB800EB216 mov  cx, cs: status_mask
1421 E4A0  in  al, fdc_stat ; read status
1423 22C5  and  al, ch
1425 32C1  xor  al, cl ; isolate what we want to poll
1427 B000  mov  al, 0
1429 7402  jz  flipy_poll_nrdy
142B B0FF  mov  al, 0Ffh

flipy_poll_nrdy:
  142D C3  ret

; ================================
; Function 14:  Sector Translate
; ================================
; Translate sector number given a translate table
; if the translate table address is 0, don't translate
; input:  CX = Sector Number
;        DX = Offset of Translate Table
; output: AX = Translated Sector Number

142E 83FA007593 cmp  dx, 0 1 jne  sectran!
1433 8BC1C3  mov  eax, cx 1 ret

sectran:
  1436 BBD9  mov  bx, cx
  1438 03DA  add  bx, dx ; add sector to table addr
  143A 8A1F  mov  bl, [bx] ; get logical sector
  143C 32Fr  xor  bh, bh
  143E 88C3  mov  ax, bx
  1440 C3  ret

; ================================
; Function 15:  Set DMA Base
; ================================
; set DMA segment

132
1441 890EA616  mov dma_seg,CX
1445 C3      ret

;==========
flushbuf:  ; Function 24: Flush Buffer
;==========
/         ; input: None
/         ; output: AL = 00h if no error occurs
/         ; = 01h if error occurs
1446 32C0  xor al,al  ; no need to flush buffer
1448 C3    ret  ; with no blocking/deblocking

;******************************************************************************
;*                       SUP/RTM CODE AREA                                    *
;******************************************************************************
1449 538BCB  rtm_poll:  ; check dev, if not ready, do rtm_poll
144C E81405B  push bx  ; mov cx,bx
1450 3CF740E  call polldev ; pop bx
1454 833E2E18FF74  cmp al,0ffh ; je pllret
145E 0E  jmp stoppoll, true ; je rtm_poll

;==========
rtm_poll2:  ; do poll_dev with no pretest
;==========
145B 8BD3B183  mov dx,bx  ; mov cl,f_polldev
145F 6997FD  jmp supif
1462 C3        pllret:  ret

;==========
polldev:  ; Function 17: Poll Device
;==========
/         ; input: CL = device number
/         ; output: AL = 000h if not ready
/         ; = 0ffh if ready
1467 B7008AD9  mov bh,0  ; mov bl,cl
1467 D1E3    shl bx,1
1469 FFA7DA14  jmp polltbl[bx]

;==========
strtlclk:  ; Function 18: Start Clock
;==========
/ Enable Flagsets on Tick Interrupts

133
; input: None
; output: None

146D C5063018FF
i472 C3
mov clockon,true
ret

;*********
stopclk:  ; Function 19: Stop Clock
;***********
; Disable Flagsets on Tick Interrupts
; input: None
; output: None

1473 C506301800
1478 C3
mov clockon,false
ret

;*********
getsegdt:  ; Function 16: Get Segment Table
;**********
; Not supported by MP/M-86
; input: None
; output: AX = 0ffffh

1479 BBFFFFFC3
mov ax,0ffffh ; ret

;*********
selmemory: ; Function 22: Select Memory
;**********
; input: None
; output: None

147D C3
ret

; idle:     ; Function 23: Idle
;==========
; input: None
; output: None
; The Idle routine is called by the Idle Process.
; Since the Idle routine has the worst priority (255)
; in the system, it will run only when nothing else
; can run. This routine cannot use any resources that
; may take it off the Ready List. This includes any
; kind of I/O that uses Poll Device or System Flags.

147E B188
1480 B876PD
call supIf
1483 BBFFFFF
jmp idle

; If all devices are Interrupt Driven then the Idle
; Routine can be the following instead:
;     halt ! jmp idle
; This cannot be used if any I/O uses POLL DEVICE
; since polling is only done during dispatches.

134
MP/M-86 System Guide

Appendix C  Example XIOS Listing

;=======
patch:
;=======

14B6 90909090909090
nop | nop | nop | nop | nop | nop
14BC 90909090909090
nop | nop | nop | nop | nop | nop
14B2 90909090909090
nop | nop | nop | nop | nop | nop
14B9 90909090909090
nop | nop | nop | nop | nop | nop
14BE 90909090909090
nop | nop | nop | nop | nop | nop
14A4 9090
nop | nop

;***************************************************
;*  SYSTEM DATA AREA
;*
;***************************************************

14A6 endcode rw 0

DSEG org (offset endcode + 1) and ff
org to an even word offset
current UDA for MPM calls

14A6 udata rw 1

;-----------------------------------------------
;   XIOS FUNCTION TABLE
;-----------------------------------------------

14A8 0112 functab dw const ; 0-console status
14AA 1912 dw conin ; 1-console input
14AC 5012 dw conout ; 2-console output
14AE 4912 dw plist ; 3-list output
14B0 6212 dw punch ; 4-punch output
14B2 6212 dw reader ; 5-reader input
14B4 8712 dw HOME ; 6-home
14B6 9F12 dw SELSDK ; 7-select disk
14B8 BE12 dw SETTRK ; 8-set track
14BA C312 dw SETSEC ; 9-set sector
14BC CB12 dw SETDMA ; 10-set DMA offset
14BE CD12 dw READ ; 11-read
14C0 1513 dw WRITE ; 12-write
14C2 6512 dw listst ; 13-list status
14C4 2E14 dw secrtn ; 14-sector translate
14C6 4114 dw setdb ; 15-set DMA base
14C8 7914 dw getsrt ; 16-get segment table
14CA 6314 dw poldev ; 17-poll device
14CC 6D14 dw strtclok ; 18-start clock
14CE 7314 dw stopclk ; 19-stop clock
14DD 7F12 dw maxconso ; 20-maximum consoles
14D2 8312 dw maxlist ; 21-max list devices
14D4 7D14 dw selemem ; 22-select memory
14D6 7614 dw idle ; 23-idle
14DA 4614 dw flushbuf ; 24-flush buffer

135
; routines to find device status
; on poll device calls

14DA 0112 polltbl dw const ;00-con 0 in
14DC 0112 dw const ;01-con 1 in
14DE 0112 dw const ;02-con 2 in
14E0 0112 dw const ;03-con 3 in
14E2 0112 dw const ;04-con 4 out
14E4 0112 dw const ;05-con 5 out
14E6 0112 dw const ;06-con 6 out
14E8 0112 dw const ;07-con 7 out
14EA 6512 dw listat ;08-list 0 out
14EC 1C14 dw flip_poll_scat ;09-dsk status

14EF 0DDA signon db cr,lf
14F0 4D5C2F4D2D38 db 'MP/M-86 V2.0 for SBC 8612', cr,lf, 0
362056322E3020666F722053424320
363631320D0h 00

150C 205566E96E69 int_trp db 'Uninitialized Interrupt', cr,lf, 0
769616C697A<56420496874
657272757074
0D0A00

1527 loc_stack rw 32 ;local stack for initialization
1567 stkBase equ offset $
1567 tickint_tcs rw 0

1567 initms rw 1
1569 initsp rw 1
156B initstack rw 32
15AB

;**********************************************************************

;*     CHARACTER I/O DATA AREA
;*     **********************************************************************

; org ((offset $) + 1) and Offset
; console i/o table for
; status mask and port address

15AC DA02 consttbl dw (c0inmask shr 8) or c0stport
15AE 4502 dw (c1inmask shr 8) or c1stport
15B0 4502 dw (c2inmask shr 8) or c2stport
15B2 4702 dw (c3inmask shr 8) or c3stport
15B4 DA01 dw (c0outmask shr 8) or c0stport
15B6 4301  dw (cloutmsk shl 8) or clstport
15B8 4501  dw (c2outmsk shl 8) or c2stport
15BA 4701  dw (c3outmsk shl 8) or c3stport
; console input table for
; poll device no. and port address
;
15BC D800  conintbl  dw (c0indev shl 8) or c0ioprt
15BE 4201  dw (c1indev shl 8) or c1ioprt
15C0 4402  dw (c2indev shl 8) or c2ioprt
15C2 4603  dw (c3indev shl 8) or c3ioprt
; console output table for
; poll device no. and port address
;
15C4 D804  conouttbl  dw (c0outdev shl 8) or c0ioprt
15C6 4205  dw (c1outdev shl 8) or c1ioprt
15C8 4406  dw (c2outdev shl 8) or c2ioprt
15CA 4607  dw (c3outdev shl 8) or c3ioprt
; list i/o table for
; status mask and port address
;
15CC 4181  loutsttbl  dw (10outmsk shl 8) or 10stport
; list output for
; poll device no. and port address
;
15CE 4008  losttbl  dw (10outdev shl 8) or 10ioprt

;*******************************************************************************
;*******************************************************************************
; DISK DATA AREA
;*******************************************************************************
;*******************************************************************************

15D0 F015F015F015 errtbl  dw er0,er1,er2,er3
F015
15D8 001610161D1E  dw er4,er5,er6,er7
2E16
15E0 411655166716  dw er8,er9,erA,erB
7C16
15E8 8C16F015F015  dw erC,erD,erE,erF
F015
15F0 0D0A4E75656C er0  db cr,lf,"Null Error ??",0
204572726F72
203F3F00
15F0  er1  equ er0
15F0  er2  equ er0
15F0  er3  equ er0
1600 0D0A436C6F63 er4  db cr,lf,"Clock Error :",0
6B204572726F
72203A00
1610 0D0A4C617465 er5  db cr,lf,"Late DMA :",0

137
db cr,lf,'ID CRC Error :',0
524320457272
6F72203A00

db cr,lf,'Data CRC Error :',0
204352432045
72726F72203A0
00

db cr,lf,'Drive Not Ready :',0
65204E6F7420
526561647920

db cr,lf,'Write Protect :',0
652050726F74
656374203A00

db cr,lf,'Trk 00 Not Found :',0
303020486F74
20466F756664
203A00

db cr,lf,'Write Fault :',0
65204661756C
74203A00

db cr,lf,'Sector Not Found :',0
60A53656374
6F7220486F74
20466F756664
203A00

erD equ er0
erE equ er0
erF equ er0

various command strings for 18271

io_com db 3 length
rd_wr db 0 read/write function code
trk db 0 track #
sect db 0 sector #

hom_com db 2,29h,0 home drive command
rd_status db 1,2ch read status command

status_mask dw 0 mask for flpy_poll

disk db 0 physical disk selected
logical to physical disk
translation table

138
MP/M-86 System Guide

Appendix C Example XIOS Listing

16B5 0001FFFF
    dtrntbl db 000h,001h,00fh,00fh
    ; E F G H
16B9 FFFFFFFF
    db 0ffh,0ffh,0ffh,0ffh
    ; I J K L
16BD FFFFFFFF
    db 0ffh,0ffh,0ffh,0ffh
    ; M N O P

;******MDISK SUPPORT*******
if mendisk
   db 002h,0ffh,0ffh,0ffh
endif

*********************
if not mendisk
   db 0ffh,0ffh,0ffh,0ffh
endif

; disk home routine table
16C5 9812
    dbkhomtbl dw flipy_home ; A drive
    ; dw flipy_home ; B drive
16C7 9812
    ;******MDISK SUPPORT*******
if mendisk
   dw hom_ret ; M drive
endif

*********************
; disk read routine table
16CB F612
    dskrdtbl dw flipy_read ; A drive
16CD F612
    dw flipy_read ; B drive

16CF D912
    ;******MDISK SUPPORT*******
if mendisk
   dw mdsk_read ; M drive
endif

*********************
; disk write routine table
16D1 3El3
    dskwrttbl dw flipy_write ; A drive
16D3 3E13
    dw flipy_write ; B drive

16D5 2:13
    ;******MDISK SUPPORT*******
if mendisk
   dw mdkw_write ; M drive
endif

*********************
include singles.lib ; read in disk definitions

= 16D7
  ; dpbase equ $ ; Base of Disk Parameter
  ; DISKS 3
=16D7 16170000
dpe0 dw x100,0000h ; Translate Table
=16DB 00000000
dw 0000h,0000h ; Scratch Area
=16DF 3F170717
dw dirbuf,dpb0 ; Dir Buff, Parm Block
=16E3 DE17BF17
dw csv0,alv0 ; Check, Alloc Vectors
=16E7 16170000
dpei dw x101,0000h ; Translate Table
=16EB 00000000
dw 0000h,0000h ; Scratch Area
=16EF 3F170717
dw dirbuf,gbbl ; Dir Buff, Parm Block
=16F3 0D18EE17
dw cv1,ai1 ; Check, Alloc Vectors
=16F7 00000000
dpe2 dw x102,0000h ; Translate Table

139
MP/M-86 System Guide

Appendix C Example XIOS Listing

+16FB 00000000 dw 0000h,0000h ;Scratch Area
+16FF 3F173017 dw dirbuf,dpb0 ;Dir Buff, Parm Block
+1703 2D181D18 dw csv2,alv2 ;Check, Alloc Vectors

= = = =
= 1944: 128 Byte Record Capacity
= 243: Kilobyte Drive Capacity
= 64: 32 Byte Directory Entries
= 64: Checked Directory Entries
= 128: Records / Extent
= 8: Records / Block
= 2: Sectors / Track
= 6: Sector Skew Factor

= = = =
= 1707 dpb0 equ offset $ ;Disk Parameter Block
= 1707 1A00 dw 26 ;Sectors Per Track
= 1708 03 db 3 ;Block Shift
= 170A 07 db 7 ;Block Mask
= 170B 00 db 0 ;Extnt Mask
= 170C F200 dw 242 ;Disk Size - 1
= 170E 3F00 dw 63 ;Directory Max
= 1710 C0 db 192 ;Alloc0
= 1711 00 db 0 ;Alloc1
= 1712 1000 dw 16 ;Check Size
= 1714 0200 dw 2 ;Offset
= 1716 xlt0 equ offset $ ;Translate Table
= 1716 0107D11 db 1,7,13,19
= 171A 19050811 db 25,5,13,17
= 171E 1703090F db 23,3,9,15
= 1722 1502080E db 21,2,8,14
= 1726 141A060C db 20,26,6,12
= 172A 1219N40A db 18,34,4,10
= 172E 1016 db 16,22
= = = =
= 001F als0 equ 31 ;Allocation Vector Size
= 0010 css0 equ 16 ;Check Vector Size
= = = =
= = = =
= Disk 1 is the same as Disk 0
= = = =
= 1707 dpbl equ dpb0 ;Equivalent Parameters
= 001F als1 equ als0 ;Same Allocation Vector
= 0010 css1 equ css0 ;Same Checksum Vector S
= 1716 xlt1 equ xlt0 ;Same Translate Table
= = = =
= = = =
= = = =
= 1016: 128 Byte Record Capacity
= 127: Kilobyte Drive Capacity
= 64: 32 Byte Directory Entries
= 0: Checked Directory Entries
= 128: Records / Extent
= 8: Records / Block
= 2: Sectors / Track
= 0: Reserved Tracks

140
1230  dpb2  equ  offset $  ;Disk Parameter Block
=1730 1AD0  dw  26  ;Sectors Per Track
=1732 03  db  3  ;Block Shift
=1733 07  db  7  ;Block Mask
=1734 00  db  0  ;Extnt Mask
=1735 7E00  dw  126  ;Disk Size - 1
=1737 3F00  dw  63  ;Directory Max
=1739 CO  db  192  ;Alloc0
=173A 00  db  0  ;Alloc1
=173B 0000  dw  0  ;Check Size
=173D 0000  dw  0  ;Offset
= 0000  xlt2  equ  0  ;No Translate Table
= 0010  als2  equ  16  ;Allocation Vector Size
= 0000  css2  equ  0  ;Check Vector Size
=

; Uninitialized Scratch Memory Follows:

= 173F  begdat  equ  offset $  ;Start of Scratch Area
=173F  dirbuf  rs  128  ;Directory Buffer
=17BF  alv0  rs  als0  ;Alloc Vector
=17DE  csv0  rs  css0  ;Check Vector
=178E  alv1  rs  als1  ;Alloc Vector
=180D  csv1  rs  css1  ;Check Vector
=181D  alv2  rs  als2  ;Alloc Vector
=182D  csv2  rs  css2  ;Check Vector
=
= 182D  enddat  equ  offset $  ;End of Scratch Area
= 00EE  datsz  equ  offset $-begdat  ;Size of Scratch Area
=182D 00  db  0  ;Marks End of Module

;********************************************************************

; SUP/RTM DATA AREA

;********************************************************************

182E 0000  stoppoll  dw  false  ;disallows poll_device

1830 00  clockon  db  false  ;if true, Tick flag is

1831 3C  tick_count  db  60  ;set on Tick interrupts

; scratch area

1832 CCCCCCCCCCCC  dw  0ccccc,0ccccc,0ccccc
1838 CCCCCCCCCCCC  dw  0ccccc,0ccccc,0ccccc
1843 CCCCCCCCCCCC  dw  0ccccc,0ccccc,0ccccc
1844 CCCCCCCCCCCC  dw  0ccccc,0ccccc,0ccccc
184A CCCCCCCCCCCC  dw  0ccccc,0ccccc,0ccccc
1850 CCCCCCCCCCCC  dw  0ccccc,0ccccc,0ccccc
1856 CCCCCCCCCCCC  dw  0ccccc,0ccccc,0ccccc

141
185C 00

db 0 ;fill last address for GENCMD
APPENDIX D

Blocking/Deblocking Algorithms

1: ;*******************************************************************************
2: ;
3: ; Sector Blocking / Deblocking
4: ;
5: ; This algorithm is a direct translation of the
6: ; CP/M-80 Version, and is included here for refer-
7: ; ence purposes only. The file DEBLOCK.LIB is in-
8: ; cluded on your MP/M-86 disk, and should be used
9: ; for actual applications. You may wish to contact
10: ; Digital Research for notices of updates.
11: ;*******************************************************************************
12: ;
13: ;*******************************************************************************
14: ; CP/M to host disk constants
15: ;
16: ; (This example is setup for MP/M block size of 16K
17: ; with a host sector size of 512 bytes, and 12 sec-
18: ; tors per track. Blksiz, hstsiz, hstsp, hstblk
19: ; and secsf may change for different hardware.)
20: ;*******************************************************************************
21: ; una equ byte ptr [BX] ;name for byte at BX
22: ;
23: ; blksiz equ 16384 ;CP/M allocation size
24: ;
25: ; hstsiz equ 512 ;host disk sector size
26: ;
27: ; hstsp equ 12 ;host disk sectors/trk
28: ;
29: ; hstblk equ hstsiz/128 ;CP/M sects/host buff
30: ;
31: ;*******************************************************************************
32: ; secsf is log2(hstblk), and is listed below for
33: ; values of hstsiz up to 2048.
34: ;*******************************************************************************
35: ;
36: ;
37: ;
38: ;
39: ;
40: ;
41: ;*******************************************************************************
42: ; secsf equ 2 ;log2(hstblk)
43: ;
44: ;*******************************************************************************
45: ;
46: ;*******************************************************************************
47: ; BDOS constants on entry to write
48: ;*******************************************************************************
49: ;
50: ;*******************************************************************************

143
51: wrall equ 0 ; write to allocated
52: wrdir equ 1 ; write to directory
53: wrsl equ 2 ; write to unallocated
54: ;
55: *****************************************************************************
56: ;
57: ; The BIOS entry points given below show the code which is relevant to deblocking only.
58: ;
59: ;
60: *****************************************************************************
61: selfdisk:
62: ; select disk
63: ; is this the first activation of the drive?
64: test DL, 1 ; lsb = 0?
65: jnz selfset
66: ; this is the first activation, clear host buff
67: mov hstact, 0
68: ;
69: selfset:
70: mov al, cl ! cbw ; put in AX
71: mov sekdsk, al ; seek disk number
72: mov cl, 4 ! shr al, cl ; times 16
73: add ax, offset dpbase
74: mov bx, ax
75: ret
76: ;
77: home:
78: ; home the selected disk
79: mov al, hstwr ; check for pending write
80: test al, al
81: jnz homed
82: ;
83: homed:
84: mov cx, 0 ; clear host active flag
85: ; (continue HOME routine)
86: ret
87: ;
88: settrk:
89: ; set track given by registers CX
90: mov sekttrk, CX ; track to seek
91: ret
92: ;
93: setsec:
94: ; set sector given by register cl
95: mov seksec, cl ; sector to seek
96: ret
97: ;
98: setdma:
99: ; set dma address given by CX
100: mov dma_off, CX
101: ret
102: ;
103: setdmb:
104: ; set segment address given by CX
105: mov dma_seg, CX

144
106:   ret
107:  ;
108:  sectran:
109:   ;translate sector number CX with table at [DX]
110:   test DX,DX   ;test for hard skewed
111:   jz notran   ;(blocked must be hard skewed)
112:   mov BX,CX
113:   add BX,DX
114:   mov BL,[BX]
115:   rst
116:  no_tran:
117:   ;hard skewed disk, physical = logical sector
118:   mov BX,CX
119:   ret
120:  ;
121:  read:
122:   ;read the selected CP/M sector
123:   mov unacnt,0   ;clear unallocated counter
124:   mov readop,1   ;read operation
125:   mov rsflag,1   ;must read data
126:   mov wrtype,wrual   ;treat as unalloc
127:   jmp rwoper   ;to perform the read
128:  ;
129:  write:
130:   ;write the selected CP/M sector
131:   mov readop,0   ;write operation
132:   mov wrtype,cr
133:   cmp cl,wrual   ;write unallocated?
134:   jnz chkuna   ;check for unalloc
135:  ;
136:  write to unallocated, set parameters
137:  ;
138:   mov unacnt,(blksz/256)   ;next unalloc recs
139:   mov al,sekdisk   ;disk to seek
140:   mov unadsk,al   ;unadsk = sekdisk
141:   mov ax,sektkr   ;unatrk = sektkr
142:   mov al,seksec   ;unasec = seksec
143:   mov unasec,al   ;
144:  ;
145:  chkuna:
146:   ;check for write to unallocated sector
147:  ;
148:  ;
149:   mov bx,offset unacnt   ;point "UNA" at UNACNT
150:   mov al,una t test al,al   ;any unalloc remain?
151:   jz alloc   ;skip if not
152:  ;
153:  more unallocated records remain
154:   dec al   ;unacnt = unacnt-1
155:  ;
156:   mov una,al   ;same disk?
157:   mov BX,offset unadsk
158:   cmp al,una   ;sekdisk = unadsk?
159:   jnz alloc   ;skip if not
160:  ;

145
disks are the same
mov AX, unatrkr
cmp AX, sektrk
;skip if not
jnz alloc
tracks are the same
;same sector?
mov al,seksec
mov BX,offset unasec
;point una at unasec
mov al,una
;seksec = unasec?
inc una
;unasec = unasec+1
mov al,una
;end of track?
cmp al,cpmsect
;count CP/M sectors
jb noovf
;skip if below
match, move to next sector for future ref
inc una
;unasec = unasec+1
mov al,una
;end of track?
cmp al,cpmsect
;count CP/M sectors
jb noovf
;skip if below
overflow to next track
mov una,0
;unasec = 0
inc unatrkr
;unatrkr=unatrkr+1
mov una,0
;unasec = 0
inc unatrkr
;unatrkr=unatrkr+1
match found, mark as unnecessary read
mov rsflag,0
;rsflag = 0
jmps twoper
;to perform the write
not an unallocated record, requires pre-read
mov unacnt,0
;unacnt = 0
mov rsflag,1
;rsflag = 1
drop through to twoper
;*****************************************************************************
* Common code for READ and WRITE follows *
*****************************************************************************
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
mov al,al
;was it already?
jz flhst
;fill host if not
host buffer active, same as seek buffer?
mov al,sekdk
216: cmp al, hstdsk
217: jnz nomatch
218:
219: ; same disk, same track?
220: mov ax, hstrtk
221: cmp ax, sektrk
222: jnz nomatch
223:
224: ; same disk, same track, same buffer?
225: mov al, sekht
226: cmp al, hstsec
227: jz match
228: ; skip if match
229: nomatch:
230: ; proper disk, but not correct sector
231: mov al, hstwrt
232: test al, al
233: jz filhst
234: ; "dirty" buffer?
235: ;no, don't need to write
236: call writehst
237: ; yes, clear host buff
238: (check errors here)
239:
240: fiilhst:
241: ; may have to fill the host buffer
242: mov al, sekdk
243: mov hstdsk, al
244: mov ax, sektrk
245: mov hstrtk, ax
246: mov al, sekht
247: mov hstsec, al
248: mov al, rsflag
249: test al, al
250: jz filhstl
251: ; need to read?
252: call readhst
253: ; yes, if 1
254: (check errors here)
255:
256: filhstl:
257: mov hstwrk, 0
258: ; no pending write
259:
260: ; match:
261: copy data to or from buffer depending on "readop"
262: mov al, seksex
263: mov hstsex, al
264: ; mask buffer number
265: and ax, secmsk
266: ; least significant bits are masked
267: mov cl, 7
268: shl ax, cl
269: ; shift left 7 (* 128 = 2**7)
270: ; ax has relative host buffer offset
271:
272: add ax, offset hstbuf
273: ; ax has buffer address
274: mov si, ax
275: ; put in source index register
276: mov di, dma_off
277: ; user buffer is dest if readop
278: ; push DS: push ES
279: ; save segment registers
280: push DS
281: push ES
282: mov ES, dma_seg
283: ; set destseg to the users seg
284: ; SI/DI and DS/ES is swapped
285: ; if write op
286: mov cx, 128/2
287: ; length of move in words
288: mov al, readop
289: test al, al
290: ; which way?
jnz rmove ;skip if read
write operation, mark and switch direction
mov hstwrt,1 ;hstwrt = 1 (dirty buffer now)
xchg zi,dl ;source/des index index
mov ax,DS
mov ES,AX ;setup DS,ES for write
mov DS,dma_seg
rmove:
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
jmp 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
*****************************************************************************
dbase equ offset $ ; disk parameter tables go here
*****************************************************************************
*****************************************************************************
Uninitialized RAM areas follow, including the
areas created by the GENDEF utility listed above.
*****************************************************************************
*****************************************************************************
seek_disk rb l ;seek disk number
seek_trk rw l ;seek track number
326: seek_seck rb 1 ; seek sector number
327:
328: hst_dsk rb 1 ; host disk number
329: hst_trk rw 1 ; host track number
330: hst_seck rb 1 ; host sector number
331:
332: seek_hst rb 1 ; seek shr secshf
333: hst_act rb 1 ; host active flag
334: hst_wrt rb 1 ; host written flag
335:
336: una_cnt rb 1 ; unalloc rec cnt
337: una_dsk rb 1 ; last unalloc disk
338: una_trk rw 1 ; last unalloc track
339: una_seck rb 1 ; last unalloc sector
340:
341: erflag rb 1 ; error reporting
342: rsflag rb 1 ; read sector flag
343: readop rb 1 ; 1 if read operation
344: wrtype rb 1 ; write operation type
345: dma_seq rw 1 ; last dma segment
346: dma_off rw 1 ; last dma offset
347: hstbuf rb hstsz ; host buffer
348: end
<table>
<thead>
<tr>
<th>INDEX</th>
</tr>
</thead>
<tbody>
<tr>
<td>B</td>
</tr>
<tr>
<td>Basic Disk Operating System, 2</td>
</tr>
<tr>
<td>BDOS, 2, 9</td>
</tr>
<tr>
<td>BIOS Jump Table, 54</td>
</tr>
<tr>
<td>blocking/deblocking algorithm, 71</td>
</tr>
<tr>
<td>Blocking/Deblocking Buffers, 46</td>
</tr>
<tr>
<td>Bootstrap ROM, 77</td>
</tr>
<tr>
<td>breakpoints, 76</td>
</tr>
<tr>
<td>C</td>
</tr>
<tr>
<td>CCB Table, 8</td>
</tr>
<tr>
<td>Character Control Block, 8</td>
</tr>
<tr>
<td>Character I/O Module, 2</td>
</tr>
<tr>
<td>checksum byte, 61</td>
</tr>
<tr>
<td>CIO, 2, 8</td>
</tr>
<tr>
<td>CMD file Header, 75</td>
</tr>
<tr>
<td>CMD Header Record, 20</td>
</tr>
<tr>
<td>Cold Start Loader, 77</td>
</tr>
<tr>
<td>CONSOLE INPUT, 10, 29</td>
</tr>
<tr>
<td>CONSOLE OUTPUT, 30</td>
</tr>
<tr>
<td>CONSOLE STATUS, 28</td>
</tr>
<tr>
<td>d</td>
</tr>
<tr>
<td>data allocation block size, 60</td>
</tr>
<tr>
<td>data block, 61, 64</td>
</tr>
<tr>
<td>Device Polling, 6</td>
</tr>
<tr>
<td>directory entries, 61</td>
</tr>
<tr>
<td>Disk Definition Tables, 57</td>
</tr>
<tr>
<td>Disk I/O Function, 36</td>
</tr>
<tr>
<td>Disk Parameter Block, 59</td>
</tr>
<tr>
<td>Disk Parameter Header, 38, 44, 57, 65</td>
</tr>
<tr>
<td>disk reset, 64</td>
</tr>
<tr>
<td>DISKDEF, 63</td>
</tr>
<tr>
<td>Dispatcher, 6, 51</td>
</tr>
<tr>
<td>E</td>
</tr>
<tr>
<td>ENTRY, 26, 53, 76</td>
</tr>
<tr>
<td>Extended Input/Output System, 2</td>
</tr>
<tr>
<td>external fragmentation, 7</td>
</tr>
<tr>
<td>F</td>
</tr>
<tr>
<td>Far Call, 26, 53</td>
</tr>
<tr>
<td>Far Return, 26, 53</td>
</tr>
<tr>
<td>fixed partition, 7</td>
</tr>
<tr>
<td>FLUSH BUFFERS, 46</td>
</tr>
<tr>
<td>G</td>
</tr>
<tr>
<td>GENDEF, 60</td>
</tr>
<tr>
<td>GENDEF Error Messages, 68</td>
</tr>
<tr>
<td>GENDEF parameters, 62</td>
</tr>
<tr>
<td>GENSYS, 1, 7, 8, 11, 15, 17, 20, 22, 23, 75</td>
</tr>
<tr>
<td>H</td>
</tr>
<tr>
<td>hardware interface, 1</td>
</tr>
<tr>
<td>HOME, 37</td>
</tr>
<tr>
<td>host disk, 72</td>
</tr>
<tr>
<td>host system, 72</td>
</tr>
<tr>
<td>I</td>
</tr>
<tr>
<td>IDLE, 51</td>
</tr>
<tr>
<td>INIT, 25, 53, 76, 79</td>
</tr>
<tr>
<td>internal fragmentation, 7</td>
</tr>
<tr>
<td>internal system calls, 71</td>
</tr>
<tr>
<td>Interrupt 224, 56</td>
</tr>
<tr>
<td>interrupt devices, 55</td>
</tr>
<tr>
<td>Interrupt Handler, 55</td>
</tr>
<tr>
<td>interrupt-driven I/O, 75</td>
</tr>
<tr>
<td>L</td>
</tr>
<tr>
<td>LDBIOS, 78</td>
</tr>
<tr>
<td>LDBIOS, 78</td>
</tr>
<tr>
<td>LDPM, 78</td>
</tr>
<tr>
<td>LIST OUTPUT, 31</td>
</tr>
<tr>
<td>LIST STATUS, 33</td>
</tr>
<tr>
<td>locked records, 21</td>
</tr>
<tr>
<td>Logical Sector Number, 44</td>
</tr>
<tr>
<td>logically invariant interface, 1</td>
</tr>
<tr>
<td>M</td>
</tr>
<tr>
<td>MAXIMUM CONSOLES, 34</td>
</tr>
<tr>
<td>MAXIMUM LIST DEVICE, 35</td>
</tr>
</tbody>
</table>
XIOS Function 12: WRITE

Returned Value:
Register AL: 0 if No Error
1 if Physical Error
ADD--> 2 Read/Only Disk

XIOS Function 24: FLUSH BUFFERS

Returned Value:
Register AL: 0 if No Error
1 if Physical Error
ADD--> 2 Read/Only Disk

The example generation of MPMLDR, although correct in itself, does not generate an MPMLDR loading at paragraph 400H as it is distributed with MP/M-86 2.0.

change:
GENCMD MPMLDR 8080 CODE[A040]

to:
GENCMD MPMLDR 8080 CODE[A0400]

in the following paragraph:

change:
... paragraph address of 040H, or byte address 0400H. ...

to:
... paragraph address of 0400H, or byte address 04000H. ...
two paragraphs further down:
change:
  byte location 0400H. ...
  byte location 84000H. ...

---

.COM files are not distributed with MP/M-86.
strike the paragraph on this page.

---

APPENDIX C - EXAMPLE XIOS LISTING

change:
  ... XIOS.A86 file provided with CP/M-86. ...
  ... XIOS.A86 file provided with MP/M-86. ...

---

NOTE:
This listing is not correct. Using ASM-86,
reassemble the file XIOS.A86 and print the
XIOS.LST file produced for a accurate listing.

---

APPENDIX D - Blocking/deblocking Algorithms

This listing is not correct. Print
the file DEBLOCK.LIB that comes with the
distribution diskettes of MP/M-86 for a
corrected version of this file. Use the
command:

   PIP PRN:=DEBLOCK.LIB