MP/M-86 T.M.

Operating System

SYSTEM GUIDE

Digital Research
P.O. Box 579
167 Central
Pacific Grove, CA 93950
(408) 649-3896
TWX 910 360 5001

COPYRIGHT

DISCLAIMER

TRADEMARKS

The "MP/M-86 SYSTEM GUIDE" was prepared using the Digital Research TEX Text Formatter.

```
**************************************
*  First Printing: September 1981   *
**************************************
```

FOREWORD

MP/M-86[T.M.] 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 distributer. 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 distributer.

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), preferrably on the target MP/M-86 machine.

TABLE OF CONTENTS

TABLE OF CONTENTS

(continued)

APPENDICES

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 (TMP). 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.

```
                        USER
                         ||
                         \/
            +--------------------+
            |   User Interface   |
            |                    |
            |       (TMP)        |
            +--------------------+
                         ||
                         \/
            +--------------------+
            |     Invariant      |
            |     Interface      |
            |                    |
            | (SUP RTM MEM CIO)  |
            |      (BDOS)        |
            +--------------------+
                         ||
                         \/
            +--------------------+
            |      Hardware      |
            |     Interface      |
            |                    |
            |      (XIOS)        |
            +--------------------+
                         ||
                         \/
                HARDWARE ENVIRONMENT
```

Figure 1-1. MP/M-86 Interface

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.

```
+---------------------------+ top of memory
|                           |
/\/\/\/\/\/\/\/\/\/\/\/\/\/\/

/\/\/\/\/\/\/\/\/\/\/\/\/\/\/
|            TPA            |
|                           |
+---------------------------+ = End of O.S.
|                           |     area
|           RSPs            |
|                           |
+---------------------------+
|        Table Area         | \
+---------------------------+ |  within
|           XIOS            |  >  64k
+---------------------------+ |
|          SYSDAT           | /
+---------------------------+
|        (XIOS Code)        | = if small
+---------------------------+    model
|        BDOS Code          |
+---------------------------+
|         CIO Code          |
+---------------------------+
|         MEM Code          |
+---------------------------+
|         RTM Code          |
+---------------------------+
|         SUP Code          |
+---------------------------+  = beginning
|                           |  of O.S. area
|            TPA            |
/\/\/\/\/\/\/\/\/\/\/\/\/\/\/

/\/\/\/\/\/\/\/\/\/\/\/\/\/\/
|                           |
+---------------------------+ 00400H
|     Interrupt Vectors     |
+---------------------------+ 00000H
```

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 (CLI) 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

Function 12:	BDOS VERSION NUMBER
Function 47:	PROGRAM CHAIN
Function 50:	CALL BIOS FUNCTION
Function 59:	PROGRAM LOAD
Function 107:	RETURN SERIAL NUMBER
Function 150:	CLI
Function 151:	CALL RPL
Function 152:	PARSE FILENAME
Function 154:	GET SYSDAT ADDRESS
Function 155:	TIME OF DAY
Function 163:	MPM VERSION NUMBER

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 time-slicing.

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 1-2. Real-Time Monitor Functions

Function 0:	SYSTEM RESET
Function 131:	POLL DEVICE
Function 132:	FLAG WAIT
Function 133:	FLAG SET
Function 134:	MAKE QUEUE
Function 135:	OPEN QUEUE
Function 136:	DELETE QUEUE
Function 137:	READ QUEUE
Function 138:	CONDITIONAL READ QUEUE
Function 139:	WRITE QUEUE
Function 140:	CONDITIONAL WRITE QUEUE
Function 141:	DELAY
Function 142:	DISPATCH
Function 143:	TERMINATE
Function 144:	CREATE PROCESS
Function 145:	SET PRIORITY
Function 156:	GET PD ADDRESS
Function 157:	ABORT SPECIFIED PROCESS

1.5 Memory Module

The Memory Management module (MEM) handles all memory functions. MP/M-86 2.0 supports an extended fixed-paritition 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 allow 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 first. 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 MFL. 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

Function 53:	GET MAXIMUM MEMORY
Function 54:	GET ABSOLUTE MAX MEMORY
Function 55:	ALLOCATE MEMORY
Function 56:	ALLOCATE ABSOLUTE MEMORY
Function 57:	FREE MEMORY
Function 58:	FREE ALL MEMORY
Function 128:	MEMORY ALLOCATION
Function 129:	MEMORY ALLOCATION
Function 130:	MEMORY FREE

Note: Functions 53 through 58 internally call Function 128 or 130. 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	160:	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/SET 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 BDOS ERROR MODE
Function 46: GET DISK FREE SPACE
Function 48: 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 devices 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.

```
      +-----+-----+-----+-----+-----+-----+-----+-----+
 00H  |     SUP ENTRY         |      RESERVED         |
      +-----+-----+-----+-----+-----+-----+-----+-----+
 08H  |                   RESERVED                    |
      +-----+-----+-----+-----+-----+-----+-----+-----+
 10H  |                   RESERVED                    |
      +-----+-----+-----+-----+-----+-----+-----+-----+
 18H  |                   RESERVED                    |
      +-----+-----+-----+-----+-----+-----+-----+-----+
 20H  |                   RESERVED                    |
      +-----+-----+-----+-----+-----+-----+-----+-----+
 28H  |     XIOS ENTRY        |      XIOS INIT        |
      +-----+-----+-----+-----+-----+-----+-----+-----+
 30H  |                   RESERVED                    |
      +-----+-----+-----+-----+-----+-----+-----+-----+
 38H  |     DISPATCHER        |        PDISP          |
      +-----+-----+-----+-----+-----+-----+-----+-----+
 40H  |   MPMSEG  |   RSPSEG  |   ENDSEG  |RESER|NCNS |
      |           |           |           | VED |     |
      +-----+-----+-----+-----+-----+-----+-----+-----+
 48H  |NLST |NCCB |  N_ | SYS_|    MMP    |  N_ | DAY |
      |     |     |FLAGS| DISK|           |SLAVE| FILE|
      +-----+-----+-----+-----+-----+-----+-----+-----+
 50H  | TEMP|TICKS|    LUL    |    CCB    |   FLAGS   |
      | DISK| /SEC|           |           |           |
      +-----+-----+-----+-----+-----+-----+-----+-----+
 58H  |   MDUL    |    MFL    |    PUL    |    QUL    |
      +-----+-----+-----+-----+-----+-----+-----+-----+
 60H  |                     QMAU                      |
      +-----+-----+-----+-----+-----+-----+-----+-----+
 68H  |    RLR    |    DLR    |    DRL    |    PLR    |
      +-----+-----+-----+-----+-----+-----+-----+-----+
 70H  | RESERVED  |  THRDRT   |    QLR    |    MAL    |
      +-----+-----+-----+-----+-----+-----+-----+-----+
 78H  |  VERSION  |  VERNUM   | MPMVERNUM |  TOD_DAY  |
      +-----+-----+-----+-----+-----+-----+-----+-----+
 80H  | TOD | TOD | TOD |NCON |NLST |NCIO | RESERVED  |
      | _HR | _MIN| _SEC| DEV | DEV | DEV |           |
      +-----+-----+-----+-----+-----+-----+-----+-----+
```

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 intermodule 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 0FFH, 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
MFL	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 though 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

NCONDEV	Number of XIOS consoles.
NLSTDEV	Number of XIOS list devices.
NCIODEV	NCONDEV + NLSTDEV
RESERVED	Reserved for Internal Use.

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 GENSYS program under an existing CP/M-86 or MP/M-86 system. GENSYS 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 GENSYS 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 (FFFF) =
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 'FFFF'
        Starting Paragraph = 2e0
                    Length = 1b0
        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
CIO.MPM	Character Input/Output Code Module
BDOS.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 MPMLDR 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.

```
A>DDT86
DDT86 1.0
rmpm.sys
 START     LENGTH
1000:0000 1000:9BFF
-d0
1000:0000 01 E2 04 08 10 E2 04 00 00 02 D2 04 EA 14 D2 04 ........
-xcs                -----                      -----
CS 0000 1008 <-------|                          |
DS 0000 14EA <-----------------------------------
SS 1C91 .
-d0
```

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 DDT86, 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 RSPs 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 XIOS 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 = 1800
                    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 partitons 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'. These 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 routines 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 initialzed 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 3-1 XIOS Register Usage

```
Entry Parameters:
          AL = function number
          CX = first parameter
          DX = second parameter
          DS = System Data Area
          ES = User Data Area

Return Codes:
          AX = return
          BX = AX
          DS = System Data Area
          ES = User Data Area
```

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 3-2 XIOS Function Numbers

Function 0	CONSOLE STATUS
Function 1	CONSOLE INPUT
Function 2	CONSOLE OUTPUT
Function 3	LIST OUTPUT
Function 4	PUNCH OUTPUT
Function 5	READER INPUT
Function 6	HOME
Function 7	SELECT DISK
Function 8	SET TRACK
Function 9	SET SECTOR
Function 10	SET DMA OFFSET
Function 11	READ
Function 12	WRITE
Function 13	LIST STATUS
Function 14	SECTOR TRANSLATE
Function 15	SET DMA BASE
Function 16	GET SEGMENT TABLE
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

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:  0ffH 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.

```
**************************************************
*                                                *
*  XIOS Function 1:  CONSOLE INPUT               *
*                                                *
**************************************************
*                                                *
*    Return Character from Specified Console     *
*                                                *
**************************************************
*  Entry Parameters:                             *
*      Register  CL:  Console number             *
*                                                *
*  Returned   Value:                             *
*      Register  AL:  Character                  *
*                BL:  Same as AL                 *
**************************************************
```

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/FLAGWAIT 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.

```
*************************************************
*                                               *
*  XIOS Function 2:  CONSOLE OUTPUT             *
*                                               *
*************************************************
*                                               *
*     Output Character to Specified Console     *
*                                               *
*************************************************
*  Entry Parameters:                            *
*      Register  CL:  Character                 *
*                DL:  Console Number            *
*                                               *
*  Returned   Value:  None                      *
*************************************************
```

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.

```
*************************************************
*                                               *
*  XIOS Function 3:  LIST OUTPUT                *
*                                               *
*************************************************
*                                               *
*   Output Character to Specified List Device   *
*                                               *
*************************************************
*  Entry Parameters:                            *
*      Register  CL:  Character                 *
*                DL:  List Device number        *
*                                               *
*  Returned   Value:  None                      *
*************************************************
```

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 MAXLIST 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                *
*                                              *
*  Returned   Value:                           *
*      Register  AL:  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, therby allowing programs to use the function as if the PUNCH device did not exist. The READER INPUT function should return a character 26 (↑Z) which indicates the end of file.

```
**************************************************
*                                                *
*  XIOS Function 13:  LIST STATUS                *
*                                                *
**************************************************
*                                                *
*          Return List Output Status             *
*                                                *
**************************************************
*  Entry Parameters:                             *
*      Register  CL:  List Device Number         *
*                                                *
*  Returned   Value:                             *
*      Register  AL:  0ffH 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 BIOS function.

```
**************************************************
*                                                *
*  XIOS Function 20:  MAXIMUM CONSOLES           *
*                                                *
**************************************************
*                                                *
*    Return the Maximum number of consoles       *
*          Supported under this XIOS             *
*                                                *
**************************************************
*  Entry Parameters:  None                       *
*                                                *
*  Returned   Value:                             *
*      Register  AL:  Number of Consoles         *
**************************************************
```

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:                           *
*      Register  AL:  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 HOME 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.

```
*************************************************
*                                               *
*   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.

```
*************************************************
*                                               *
*  XIOS Function 7:  SELECT DISK                *
*                                               *
*************************************************
*                                               *
*       Select the specified Disk Drive         *
*                                               *
*************************************************
*  Entry Parameters:                            *
*      Register  CL:  Disk Drive Number         *
*                DL:(bit 0):0 if first select   *
*                                               *
*  Returned   Value:  Offset of DPH             *
*      Register  AX:  Offset of DPH             *
*                BX:  Same as AX                *
*************************************************
```

The SELECT DISK (SELDSK) 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, SELDSK returns 0000H as an error indicator. Although SELDSK must return the Header address on each call, it is advisable to postpone the actual physical disk select operation until an I/O function (seek, read or write) is performed. This is due to the fact that disk select operations may take place without a subsequent disk operation and thus disk access may be substantially slower using some disk controllers.

On entry to SELDSK it is possible to determine whether it is the first time the specified disk has been selected. Register DL, bit 0 (least significant bit) is a zero if the drive has not been previously selected. This information is of interest in systems 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.

```
***************************************************
*                                                 *
*  XIOS Function 9:  SET SECTOR                   *
*                                                 *
***************************************************
*                                                 *
*         Set Specified Sector Number             *
*                                                 *
***************************************************
*  Entry Parameters:                              *
*      Register  CX:  Sector Number               *
*                                                 *
*  Returned   Value:  None                        *
***************************************************
```

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 10:  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 0FFH 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 128-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 occured

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, SET 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-sectored 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  *
*                                               *
* Returned  Value:                              *
*     Register  AX:  Physical Sector Number     *
*               BX:  Same as BX                 *
*************************************************
```

The SECTOR TRANSLATE (SECTRAN) 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, SECTRAN receives a Logical Sector Number. The Logical Sector Number may range from 0 to the number of sectors -1. SECTRAN 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, SECTRAN computes and returns the translated Sector Number. SECTRAN 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.

```
*************************************************
*                                               *
*  XIOS Function 24:  FLUSH BUFFERS             *
*                                               *
*************************************************
*                                               *
*    Write all pending write buffers to disk    *
*                                               *
*************************************************
*  Entry Parameters:  None                      *
*                                               *
*    Returned Value:                            *
*       Register AL:  0 if No Error             *
*                     1 if Physical Error       *
*************************************************
```

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

```
*************************************************
*                                               *
*  XIOS Function 17:  POLL DEVICE               *
*                                               *
*************************************************
*                                               *
*    Poll Specified Device and Return Status    *
*                                               *
*************************************************
*  Entry Parameters:                            *
*      Register  CL:  Poll Device Number        *
*                                               *
*  Returned   Value:                            *
*       Register AL:  0ffH 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 a 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.

```
*************************************************
*                                               *
*  XIOS Function 18:  START CLOCK               *
*                                               *
*************************************************
*                                               *
*           Turn on Tick Flag Setting           *
*                                               *
*************************************************
*  Entry Parameters:  None                      *
*                                               *
*  Returned   Value:  None                      *
*************************************************
```

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                *
*                                               *
*************************************************
*                                               *
*             Disable 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 1000H 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 GENSYS:

(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-68 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 instuctions 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 8080 environment due to machine architecture differences. The example TICK 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 interupting 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 4-1. Disk Parameter Header Elements

Element	Description
XLT	Offset of the logical-to-physical Translation Vector, if used for this particular drive, or the value 0000H if no sector translation takes place (i.e, the Physical and Logical Sector Numbers are the same). Disk drives with identical Sector Skew Factors share the same Tranlation Vector.
0000	Scratchpad values for use within the BDOS (initial value is unimportant).
DIRBUF	Offset of a 128-byte scratchpad area for directory operations within BDOS. All DPHs address the same scratchpad area.
DPB	Offset of a Disk Parameter Block for this drive. Drives with identical disk characteristics address the same Disk Parameter Block.
CSV	Offset of a scratchpad area used for software check for changed disks. This offset is different for each DPH.

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

Given n disk drives, the 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:

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

Figure 4-2. DPH Table

where the label DPBASE 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,NDISKS ;N BEYOND MAX DISKS?
          JNB     RETURN    ;RETURN IF SO
                            ;0 <= N < NDISKS
          MOV     CH,0      ;DOUBLE (N)
          MOV     BX,CX     ;BX = N
          MOV     CL,4      ;READY FOR * 16
          SHL     BX,CL     ;N = N * 16
          MOV     CX,OFFSET DPBASE
          ADD     BX,CX     ;DPBASE + N * 16
RETURN:   RET               ;BX - .DPH (N)
```

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

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

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

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

Field	Definition
SPT	is the total number of sectors per track
BSH	is the data allocation Block Shift Factor, determined by the data block allocation size (BLS).
BLM	is the Block Mask which is also determined by the data block allocation size (BLS).
EXM	is the Extent Mask, determined by the data block allocation size and the number of disk blocks.
DSM	determines the total storage capacity of the disk drive.
DRM	determines the total number of directory entries that can be stored on this drive.
AL0,AL1	determine reserved directory blocks.
CKS	is the size of the directory checksum vector. If the high-order bit of CKS is on (i.e. > 8000H), 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.
OFF	is the number of reserved tracks at the beginning of the (logical) disk.

Although these table values are produced automatically by GENDEF, it is worthwhile reviewing the derivation of each field so that the values may be cross-checked when necessary. The values of BSH and BLM determine (implicitly) the data 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 4-3. BSH and BLM Values for Selected BLS

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

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

Table 4-4. Maximum EXM Values

BLS	DSM < 256	DSM > 255
1,024	0	N/A
2,048	1	0
4,096	3	1
8,192	7	3
16,384	15	7

The value of DSM is the maximum data block number supported by this particular drive, measured in BLS units. The product BLS times (DSM+1) is the total number of bytes held by the drive and 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 4-5. BLS and Number of Directory Entries

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

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

The CKS value is determined as follows: if the disk drive media is removable, then CKS = (DRM+1)/4, where DRM is the last directory entry number. If the media is fixed, then set CKS = 0 (no directory records are checked in this case).

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

To complete the discussion of the DPB, recall that several 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 BDOS 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 DPB.

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 4-6. GENDEF Optional Parameters

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

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

```
DPBASE  EQU  $
```

which requires a MOV CX,OFFSET DPBASE in the SELDSK subroutine. 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 8086/8088 mode in order to generate tables acceptable for operation with Z80, 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 and 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 bls units
dir    is the number of directory entries
cks    is the number of "checked" directory entries
ofs    is the track offset to logical track 00
[0]    is an optional 1.4 compatibility flag
```

The value "dn" is the drive number being defined with this DISKDEF statement. The "fsc" parameter accounts for differing sector numbering systems, and is usually 0 or 1. The "lsc" is the last numbered sector on a track. When present, the "skf" parameter defines the Sector Skew Factor 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 "bls" parameter specifies the number of bytes allocated to each data block, and takes on the values 1024, 2048, 4096, 8192, or 16384. Generally, performance increases with larger data block sizes because there are fewer directory references. Also, logically connected data records are physically close on the disk. Further, each directory entry addresses more data and the amount of BIOS work space is reduced. The "dks" specifies the total disk size in "bls" units. That is, if the bls = 2048 and dks = 1000, then the total disk capacity is 2,048,000 bytes. If "dks" is greater than 255, then the block size parameter "bls" must be greater than 1024. The value of "dir" is the total number of directory entries which may exceed 255, if desired.

The "cks" parameter determines the number of directory items to check on each directory scan, and is used internally to detect changed disks during system operation, where an intervening 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 243K byte disk capacity, 64 checked directory entries, and two Operating System tracks.

The DISKS statement generates n 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 DISKS statement generates a table of the form:

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

where the DPH labels are included for reference purposes to show the beginning table addresses for each drive 0 through 3. The values contained within the Disk Parameter Header are described 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)
1DB0 =      ENDDAT EQU OFFSET $
013C =      DATSIZ EQU OFFSET $-BEGDAT
```

which indicates that uninitialized RAM begins at offset 1C72H, ends at 1DB0H-1, and occupies 013CH bytes. 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 $C parameter to GENCMD will match the output from STAT. The STAT command form

```
STAT d:DSK:
```

decodes the Disk Parameter Block for drive d (d=A,...,P) and displays the values shown below:

```
r: 128-Byte Record Capacity
k: Kilobyte Drive  Capacity
d: 32-Byte Directory Entries
c: Checked  Directory Entries
e: Records/ Extent
b: Records/ Block
s: Sectors/ Track
t: Reserved Tracks
```

4.5.3 GENDEF Output

GENDEF produces a listing of the statements included in the DEF file at the user console (↑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

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

Table 4-8. GENDEF Input and Output Error Messages

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

Given the file TWO.DEF containing the following statements

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

the command

```
gencmd two $c
```

produces the console output

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

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

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

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

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, hsttrk 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 hstbuf of length hstsiz. 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 'M' 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 necesary 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 2,1,26,,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-86 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-86 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 080H-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 BASE 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

and 2CH for INIT. If an 8080 Model, is used these functions will be at offset 1003H and 1000H 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 0H in the Code Segment. Break points can then be set within the XIOS for debugging.

5.2 Running under MP/M-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:

```
o The 86/12 Bootstrap ROM (BOOT ROM)
o The Cold Start Loader    (MPMLDR)
o 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 0ff000H, and receives control when the system reset button is depressed. In a non-standard environment, the BOOT ROM is replaced by an equivalent initial loader and, therefore, the ROM itself is not included with 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 MPMLDR into memory from the first two system tracks and then passes program control to the MPMLDR for execution.

6.1 The Cold Start Load Operation

The MPMLDR 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 MPMLDR 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 |
+------+----------+---------+---------+---------+
   8b      16b        16b       16b       16b
```

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 (LDBDOS), 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:

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

Figure 6-2. MPMLDR Organization

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 LDMPM 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 LDMPM program at byte offset 0003H. The LDMPM module opens the MPM.SYS file, loads the MP/M-86 system into memory and transfers control to MP/M-86 through the JMPF MPM instruction at the end of LDMPM execution, thus completing the cold start sequence.

The files LDMPM.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 0003H at the end of the INIT subroutine. Use ASM-86 to assemble the LDBIOS.A86 program:

```
ASM86 LDBIOS
```

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

```
PIP MPMLDR.H86=LDMPM.H86,LDBDOS.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 GENCMD utility:

```
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 0400H. 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.

```
LDCOPY MPMLDR
```

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

```
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 128-byte Header Record similar to the MPMLDR program:

```
+------+----------+---------+---------+---------+
|G-Form| G-Length |  A-Base |  G-Min  |  G-Max  |
+------+----------+---------+---------+---------+
| lor2 | xxxxxxxx |  1008   | xxxxxxx | xxxxxxx |
+------+----------+---------+---------+---------+
   8b      16b        16b       16b       16b
```

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 load address at paragraph 01008H, or byte address 10080H. 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.

```
              +----+----+-+--------+
              |GD#1|GD#2|0|////////|
              +----+----+-+--------+ = Beginning of O.S.
CS:0000H      |     SUP code       |
              |     RTM code       |
              |     MEM code       |
              |     CIO code       |
              |    BDOS code       |
              |   (XIOS code)      | = if Small Model
              +--------------------+
DS:0000H      |      SYSDAT        |
              |       XIOS         |
              |    Table area      |
              +--------------------+
              |                    |
              |       RSPs         |
              |                    |
              +--------------------+ = End of O.S.
```

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.DEF statements shown below:

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

The MPM.SYS file is read by the MPMLDR program beginning at the address given by A-Base (byte address 10080H), 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 recomended 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.A86 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.H86 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-80 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*
* this is the original BOOT ROM listing 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 loca-  *
;* tion 00000H, then initializes the segment*
;* registers and the stack pointer.  The    *
;* various peripheral interface chips on the*
;* SBC 86/12 are initialized.  The 8251     *
;* serial interface is configured for a 9600*
;* baud asynchronous terminal, and the in-  *
;* terrupt controller is setup for inter-   *
;* rupts 10H-17H (vectors at 00040H-0005FH) *
;* and edge-triggered auto-EOI (end of in-  *
;* terrupt) mode with all interrupt levels  *
;* masked-off.  Next, the SBC 204 Diskette  *
;* controller is initialized, and track 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       *
;* 0FF000H on the SBC 86/12 CPU board.  ROM *
;* 0 contains the even memory locations, and*
;* ROM 1 contains the odd addresses.  BOOT  *
;* ROM uses RAM between 00000H and 000FFH   *
;* (absolute) for a scratch area, along with*
;* the sector 1 buffer.                     *
;*********************************************
```

```
 00FF          true            equ     0ffh
 FF00          false           equ     not true
               ;
 FF00          debug           equ     false
               ;debug = true indicates bootstrap is in same roms
               ;with SBC 957 "Execution Vehicle" monitor
               ;at FE00:0 instead of FF00:0
               ;
 000D          cr              equ     13
 000A          lf              equ     10
               ;
               ;       disk ports and commands
               ;
 00A0          base204         equ     0a0h
 00A0          fdccom          equ     base204+0
 00A0          fdcstat         equ     base204+0
 00A1          fdcparm         equ     base204+1
 00A1          fdcrslt         equ     base204+1
 00A2          fdcrst          equ     base204+2
 00A4          dmacadr         equ     base204+4
 00A5          dmaccont        equ     base204+5
 00A6          dmacscan        equ     base204+6
 00A7          dmacsadr        equ     base204+7
 00A8          dmacmode        equ     base204+8
 00A8          dmacstat        equ     base204+8
 00A9          fdcsel          equ     base204+9
 00AA          fdcsegment      equ     base204+10
 00AF          reset204        equ     base204+15
               ;
               ;actual console baud rate
 2580          baud_rate       equ     9600
               ;value for 8253 baud counter
 0008          baud            equ     768/(baud_rate/100)
               ;
 00DA          csts            equ     0DAh    ;i8251 status port
 00D8          cdata           equ     0D8h    ; "     data port
               ;
 00D0          tch0            equ     0D0h    ;8253 PIC channel 0 por
 00D2          tch1            equ     tch0+2  ;ch 1 port
 00D4          tch2            equ     tch0+4  ;ch 2 port
 00D6          tcmd            equ     tch0+6  ;8253 command port
               ;
 00C0          icp1            equ     0C0h    ;8259a port 0
 00C2          icp2            equ     0C2h    ;8259a port 1
               ;
               ;
                       IF NOT DEBUG
 FF00          ROMSEG          EQU     0FF00H  ;normal
                       ENDIF
               ;
                       IF DEBUG                ;share prom with SBC 95
               ROMSEG          EQU     0FE00H
                       ENDIF
               ;
```

```
                        ;
                        ;       This long jump prom'd in by hand
                        ;       cseg    0ffffh          ;reset goes to here (ff
                        ;       JMPF    BOTTOM          ;boot is at bottom of P
                        ;       EA 00 00 00 FF          ;cs = bottom of prom (f
                        ;                                        ip = 0
                        ;       EVEN PROM       ODD PROM
                        ;       7F8 - EA        7F8 - 00
                        ;       7F9 - 00        7F9 - 00
                        ;       7FA - FF                ;this is not done if sp
                        ;
  FF00                          cseg    romseg
                        ;
                        ;First, move our data area into RAM at 0000:0200
                        ;
0000 8CC8                       mov ax,cs
0002 8ED8                       mov ds,ax       ;point DS to CS for source
0004 BE5C01                     mov SI,drombegin        ;start of data
0007 BF0002                     mov DI,offset ram_start ;offset of destination
000A B80000                     mov ax,0
000D 8EC0                       mov es,ax       ;destination segment is 0000
000F B9E600                     mov CX,data_length      ;how much to move in by
0012 F3A4                       rep movs al,al          ;move out of eprom a by
                        ;
0014 B80000                     mov ax,0
0017 8ED8                       mov ds,ax       ;data segment now in RAM
0019 8ED0                       mov ss,ax
001B BC2A03                     mov sp,stack_offset     ;Initialize stack segme
001E FC                         cld                     ;clear the direction fl
                        ;
                                IF NOT DEBUG
                        ;
                        ;Now, initialize the console USART and baud rate
                        ;
001F B00E                       mov al,0Eh
0021 E6DA                       out csts,al     ;give 8251 dummy mode
0023 B040                       mov al,40h
0025 E6DA                       out csts,al     ;reset 8251 to accept mode
0027 B04E                       mov al,4Eh
0029 E6DA                       out csts,al     ;normal 8 bit asynch mode, * 16
002B B037                       mov al,37h
002D E6DA                       out csts,al     ;enable Tx & Rx
002F B0B6                       mov al,0B6h
0031 E6D6                       out tcmd,al     ;8253 ch.2 square wave mode
0033 B80800                     mov ax,baud
0036 E6D4                       out tch2,al     ;low of the baud rate
0038 8AC4                       mov al,ah
003A E6D4                       out tch2,al     ;high of the baud rate
                        ;
                                ENDIF
                        ;
                        ;Setup the 8259 Programmable Interrupt Controller
                        ;
003C B013                       mov al,13h
```

```
003E E6C0                 out icp1,al      ;8259a ICW 1  8086 mode
0040 B010                 mov al,10h
0042 E6C2                 out icp2,al      ;8259a ICW 2  vector @ 40-5F
0044 B01F                 mov al,1Fh
0046 E6C2                 out icp2,al      ;8259a ICW 4  auto EOI master
0048 B0FF                 mov al,0FFh
004A E6C2                 out icp2,al      ;8259a OCW 1  mask all levels o
                  ;
                  ;Reset and initialize the iSBC 204 Diskette Interface
                  ;
                  restart:        ;also come back here on fatal errors
004C E6AF                 out reset204,AL ;reset iSBC 204 logic and
004E B001                 mov AL,1
0050 E6A2                 out fdcrst,AL    ;give 8271 FDC
0052 B000                 mov al,0
0054 E6A2                 out fdcrst,AL    ;  a reset command
0056 BB1502               mov BX,offset specs1
0059 E8E100               CALL sendcom   ;program
005C BB1B02               mov BX,offset specs2
005F E8DB00               CALL sendcom   ;  Shugart SA-800 drive
0062 BB2102               mov BX,offset specs3
0065 E8D500               call sendcom   ;    characteristics
0068 BB1002       homer:  mov BX,offset home
006B E85800               CALL execute     ;home drive 0
                  ;
006E BB2A03               mov bx,sector1  ;offset for first sector DMA
0071 B80000               mov ax,0
0074 8EC0                 mov es,ax        ;segment "    "    "    "
0076 E8A700               call setup_dma
                  ;
0079 BB0202               mov bx,offset read0
007C E84700               call execute     ;get T0 S1
                  ;
007F 8E062D03             mov es,ABS
0083 BB0000               mov bx,0         ;get loader load address
0086 E89700               call setup_dma  ;setup DMA to read loader
                  ;
0089 BB0602               mov bx,offset read1
008C E83700               call execute     ;read track 0
008F BB0B02               mov bx,offset read2
0092 E83100               call execute     ;read track 1
                  ;
0095 8C06E802             mov leap_segment,ES
                  ;       setup far jump vector
0099 C706E6020000         mov leap_offset,0
                  ;
                  ;       enter LOADER
009F FF2EE602             jmpf dword ptr leap_offset
                  ;
                  pmsg:
00A3 8A0F                 mov cl,[BX]
00A5 84C9                 test cl,cl
00A7 7476                 jz return
00A9 E80400               call conout
```

```
00AC 43                 inc BX
00AD E9F3FF             jmp pmsg
              ;
              conout:
00B0 E4DA               in al,csts
00B2 A801               test al,1
00B4 74FA               jz conout
00B6 8AC1               mov al,cl
00B8 E6D8               out cdata,al
00BA C3                 ret
              ;
              conin:
00BB E4DA               in al,csts
00BD A802               test al,2
00BF 74FA               jz conin
00C1 E4D8               in al,cdata
00C3 247F               and al,7Fh
00C5 C3                 ret
              ;
              ;
              ;
              execute:          ;execute command string @ [BX]
                                ;<BX> points to length,
                                ;followed by Command byte
                                ;followed by length-1 parameter bytes
              ;
00C6 891E0002           mov     lastcom,BX      ;remember what it was
              retry:                            ;retry if not ready d
00CA E87000             call    sendcom         ;execute the command
                                                ;now, let's see what t
                                                ;of status poll was ne
                                                ;for that command type
00CD 8B1E0002           mov     BX,lastcom      ;point to command stri
00D1 8A4701             mov     AL,1[BX]        ;get command op code
00D4 243F               and     AL,3fh          ;drop drive code bits
00D6 B90008             mov     CX,0800h        ;mask if it will be "i
00D9 3C2C               cmp     AL,2ch          ;see if interrupt type
00DB 720B               jb      execpoll
00DD B98080             mov     CX,8080h        ;else we use "not comm
00E0 240F               and     AL,0fh          ;unless . . .
00E2 3C0C               cmp     AL,0ch          ;there isn't
00E4 B000               mov AL,0
00E6 7737               ja return               ;any result at all
              ;
              execpoll:         ;poll for bit in b, toggled with c
00E8 E4A0               in AL,FDCSTAT
00EA 22C5               and AL,CH
00EC 32C174F8           xor AL,CL ! JZ execpoll
              ;
00F0 E4A1               in      AL,fdcrslt      ;get result register
00F2 241E               and     AL,1eh          ;look only at result t
00F4 7429               jz      return          ;zero means it was a g
              ;
00F6 3C10               cmp al,10h
```

```
00F8 7513               jne fatal               ;if other than "Not Re
                ;
00FA BB1302             mov bx,offset rdstat
00FD E83D00             call sendcom            ;perform read status co
                rd_poll:
0100 E4A0               in al,fdc_stat
0102 A880               test al,80h             ;wait for command not b
0104 75FA               jnz rd_poll
0106 8B1E0002           mov bx,last_com         ;recover last attempted
010A E9BDFF             jmp retry               ;and try it over again
                ;
                fatal:                          ; fatal error
010D B400               mov ah,0
010F 8BD8               mov bx,ax               ;make 16 bits
0111 8B9F2702           mov bx,errtbl[BX]
                ;       print appropriate error message
0115 E88BFF             call pmsg
0118 E8A0FF             call conin              ;wait for key strike
011B 58                 pop ax                  ;discard unused item
011C E92DFF             jmp restart             ;then start all over
                ;
                return:
011F C3                 RET                     ;return from EXECUTE
                ;
                setupdma:
0120 B004               mov AL,04h
0122 E6A8               out dmacmode,AL         ;enable dmac
0124 B000               mov al,0
0126 E6A5               out dmaccont,AL         ;set first (dummy) byte
0128 B040               mov AL,40h
012A E6A5               out dmaccont,AL         ;force read data mode
012C 8CC0               mov AX,ES
012E E6AA               out fdcsegment,AL
0130 8AC4               mov AL,AH
0132 E6AA               out fdcsegment,AL
0134 8BC3               mov AX,BX
0136 E6A4               out dmacadr,AL
0138 8AC4               mov AL,AH
013A E6A4               out dmacadr,AL
013C C3                 RET
                ;
                ;
                ;
                sendcom:        ;routine to send a command string to SB
013D E4A0               in AL,fdcstat
013F 2480               and AL,80h
0141 75FA               jnz sendcom     ;insure command not busy
0143 8A0F               mov CL,[BX]     ;get count
0145 43                 inc BX
0146 8A07               mov al,[BX]     ;point to and fetch command byt
0148 E6A0               out fdccom,AL   ;send command
                parmloop:
014A FEC9               dec CL
014C 74D1               jz return       ;see if any (more) parameters
```

```
014E 43                     inc BX              ;point to next parameter
                    parmpoll:
014F E4A0                   in AL,fdcstat
0151 2420                   and AL,20h
0153 75FA                   jnz parmpoll        ;loop until parm not full
0155 8A07                   mov AL,[BX]
0157 E6A1                   out fdcparm,AL      ;output next parameter
0159 E9EEFF                 jmp parmloop        ;go see about another
                    ;
                    ;
                    ;       Image of data to be moved to RAM
                    ;
  015C              drombegin equ offset $
                    ;
015C 0000           clastcom        dw      0000h   ;last command
                    ;
015E 03             creadstring     db      3       ;length
015F 52                             db      52h     ;read function code f
0160 00                             db      0       ;track #
0161 01                             db      1       ;sector #
                    ;
0162 04             creadtrk0       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             creadtrk1       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 026900         chome0          db      2,69h,0
016F 016C           crdstat0        db      1,6ch
0171 05350D         cspecs1         db      5,35h,0dh
0174 0808E9                         db      08h,08h,0e9h
0177 053510         cspecs2         db      5,35h,10h
017A FFFFFF                         db      255,255,255
017D 053518         cspecs3         db      5,35h,18h
0180 FFFFFF                         db      255,255,255
                    ;
0183 4702           cerrtbl dw      offset er0
0185 4702                   dw      offset er1
0187 4702                   dw      offset er2
0189 4702                   dw      offset er3
018B 5702                   dw      offset er4
018D 6502                   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 C502                   dw      offset erB
```

```
019B D302                  dw      offset erC
019D 4702                  dw      offset erD
019F 4702                  dw      offset erE
01A1 4702                  dw      offset erF
                      ;
01A3 0D0A4E756C6C Cer0     db      cr,lf,'Null Error ??',0
     204572726F72
     203F3F00
  01A3                Cer1     equ     cer0
  01A3                Cer2     equ     cer0
  01A3                Cer3     equ     cer0
01B3 0D0A436C6F63 Cer4     db      cr,lf,'Clock Error',0
     6B204572726F
     7200
01C1 0D0A4C617465 Cer5     db      cr,lf,'Late DMA',0
     20444D4100
01CC 0D0A49442043 Cer6     db      cr,lf,'ID CRC Error',0
     524320457272
     6F7200
01DB 0D0A44617461 Cer7     db      cr,lf,'Data CRC Error',0
     204352432045
     72726F7200
01EC 0D0A44726976 Cer8     db      cr,lf,'Drive Not Ready',0
     65204E6F7420
     526561647900
01FE 0D0A57726974 Cer9     db      cr,lf,'Write Protect',0
     652050726F74
     65637400
020E 0D0A54726B20 CerA     db      cr,lf,'Trk 00 Not Found',0
     3030204E6F74
     20466F756E64
     00
0221 0D0A57726974 CerB     db      cr,lf,'Write Fault',0
     65204661756C
     7400
022F 0D0A53656374 CerC     db      cr,lf,'Sector Not Found',0
     6F72204E6F74
     20466F756E64
     00
  01A3                CerD     equ     cer0
  01A3                CerE     equ     cer0
  01A3                CerF     equ     cer0
                      ;
  0242                dromend equ offset $
                      ;
  00E6                data_length      equ dromend-drombegin
                      ;
                      ;         reserve space in RAM for data area
                      ;         (no hex records generated here)
                      ;
  0000                          dseg     0
                                org      0200h
                      ;
  0200                ram_start         equ      $
```

```
0200            lastcom         rw      1           ;last command
0202            read0           rb      4           ;read track 0 sector 1
0206            read1           rb      5           ;read T0 S2-26
020B            read2           rb      5           ;read T1 S1-26
0210            home            rb      3           ;home drive 0
0213            rdstat          rb      2           ;read status
0215            specs1          rb      6
021B            specs2          rb      6
0221            specs3          rb      6
0227            errtbl          rw      16
0247            er0             rb      length cer0     ;16
  0247          er1             equ     er0
  0247          er2             equ     er0
  0247          er3             equ     er0
0257            er4             rb      length cer4     ;14
0265            er5             rb      length cer5     ;11
0270            er6             rb      length cer6     ;15
027F            er7             rb      length cer7     ;17
0290            er8             rb      length cer8     ;18
02A2            er9             rb      length cer9     ;16
02B2            erA             rb      length cerA     ;19
02C5            erB             rb      length cerB     ;14
02D3            erC             rb      length cerC     ;19
  0247          erD             equ     er0
  0247          erE             equ     er0
  0247          erF             equ     er0
                ;
02E6            leap_offset     rw      1
02E8            leap_segment    rw      1
                ;
                ;
02EA                            rw      32          ;local stack
  032A          stack_offset    equ     offset $;stack from here down
                ;
                ;               T0 S1 read in here
  032A          sector1         equ offset $
                ;
032A            Ty              rb      1
032B            Len             rw      1
032D            Abs             rw      1           ;ABS is all we care abo
032F            Min             rw      1
0331            Max             rw      1
                                end
```

APPENDIX B

LDBIOS 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 (LDBIOS) *
                ;* for LDMPM Configured for iSBC 86/12 with  *
                ;* the iSBC 204 Floppy Disk Controller       *
                ;*                                           *
                ;* The only modifications of the CP/M-86     *
                ;* LDBIOS for this MP/M-86 LDBIOS 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/NET for the 8086 or 8088 Micro-
                ;       processor)

FFFF            true            equ -1
0000            false           equ not true

                ;*********************************************
```

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

 00E0            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
 00D8            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+0    ;8271 in status
 00A1            fdc_parm        equ base204+1    ;8271 out parameter
 00A1            fdc_rslt        equ base204+1    ;8271 in result
 00A2            fdc_rst         equ base204+2    ;8271 out reset
 00A4            dmac_adr        equ base204+4    ;8257 DMA base addr
 00A5            dmac_cont       equ base204+5    ;8257 out control
 00A6            dmac_scan       equ base204+6    ;8257 out scan cont
 00A7            dmac_sadr       equ base204+7    ;8257 out scan addr
 00A8            dmac_mode       equ base204+8    ;8257 out mode
 00A8            dmac_stat       equ base204+8    ;8257 in status
 00A9            fdc_sel         equ base204+9    ;FDC select port (n
 00AA            fdc_segment     equ base204+10   ;segment address re
 00AF            reset_204       equ base204+15   ;reset entire inter

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

                         cseg
                         org     ccpoffset
                 ccp:
                         org     bios_code

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

1200 E93C00       jmp INIT        ;Enter from BOOT ROM or LOADER
```

```
1203 E95B00          jmp WBOOT       ;Arrive here from BDOS 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 E97000          jmp LISTOUT     ;write character to list device
1212 E96E00          jmp PUNCH       ;write character to punch device
1215 E96B00          jmp READER      ;return char from reader device
1218 E9B600          jmp HOME        ;move to trk 00 on cur sel drive
121B E99200          jmp SELDSK      ;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 E9DB00          jmp READ        ;read a 128 byte sector
122A E9DC00          jmp WRITE       ;write a 128 byte sector
122D E95200          jmp LISTST      ;return list status
1230 E9BD00          jmp SECTRAN     ;xlate logical->physical sector
1233 E9C600          jmp SETDMAB     ;set seg base for buff (DMA)
1236 E9C800          jmp GETSEGT     ;return offset of Mem Desc Table
1239 E94A00          jmp GETIOBF     ;return I/O map byte (IOBYTE)
123C E94A00          jmp SETIOBF     ;set I/O map byte (IOBYTE)

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

                    INIT:   ;initialize hardware
123F 8CC8                   mov ax,cs       ;we entered with a JMPF so
1241 8ED0                   mov ss,ax       ; CS: as the initial value
1243 8ED8                   mov ds,ax       ;       DS:,
1245 8EC0                   mov es,ax       ;       and ES:
                            ;use local stack during initialization
1247 BC8A16                 mov sp,offset stkbase
124A FC                     cld             ;set forward direction

124B 1E                     push ds         ;save data segment
124C B80000                 mov ax,0
124F 8ED8                   mov ds,ax       ;point to segment zero
                            ;BDOS interrupt offset
1251 C70680030604           mov bdos_offset,bdos_ofst
1257 8C0E8203               mov bdos_segment,CS ;bdos interrupt segment
125B 1F                     pop ds          ;restore data segment

125C B100                   mov cl,0        ;default to dr A: on coldst
125E E9A2EE                 jmp ccp         ;jump to cold start entry o

1261 E9A5EE         WBOOT:  jmp ccp+6       ;direct entry to CCP at com

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

```
                    ;*                                               *
                    ;*************************************************

                    CONST:              ;console status
1264 E4DA                   in al,csts
1266 2402                   and al,2
1268 7402                   jz const_ret
126A 0CFF                   or al,255           ;return non-zero if RDA
                    const_ret:
126C C3                     ret                 ;Receiver Data Available

                    CONIN:                      ;console input
126D E8F4FF                 call const
1270 74FB                   jz CONIN            ;wait for RDA
1272 E4D8                   in al,cdata
1274 247F                   and al,7fh          ;read data and remove parit
1276 C3                     ret

                    CONOUT:             ;console output
1277 E4DA                   in al,csts
1279 2401                   and al,1            ;get console status
127B 74FA                   jz CONOUT           ;wait for TBE
127D 8AC1                   mov al,cl
127F E6D8                   out cdata,al        ;Transmitter Buffer Empty
1281 C3                     ret                 ;then return data

                    LISTOUT:                    ;list device output
                    LISTST:                     ;poll list status

1282 C3                     ret

                    PUNCH:  ;not implemented in this configuration
                    READER:
1283 B01A                   mov al,1ah
1285 C3                     ret                 ;return EOF for now

                    GETIOBF:
1286 B000                   mov al,0            ;TTY: for consistency
1288 C3                     ret                 ;IOBYTE not implemented

                    SETIOBF:
1289 C3                     ret                 ;iobyte not implemented

                    zero_ret:
128A 2400                   and al,0
128C C3                     ret                 ;return zero in AL and flag

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

                    uconecho:
128D E8DDFF                 call CONIN          ;get a console character
1290 50                     push ax
1291 8AC8                   mov cl,al           ;save and
1293 E8E1FF                 call CONOUT
```

```
 1296 58                 pop ax          ;echo to console
 1297 3C61               cmp al,'a'
 1299 7206               jb uret         ;less than 'a' is ok
 129B 3C7A               cmp al,'z'
 129D 7702               ja uret         ;greater than 'z' is ok
 129F 2C20               sub al,'a'-'A'  ;else shift to caps
                 uret:
 12A1 C3                 ret

                 ;       utility subroutine to print messages

                 pmsg:
 12A2 8A07               mov al,[BX]     ;get next char from message
 12A4 84C0               test al,al
 12A6 7428               jz return       ;if zero return
 12A8 8AC8               mov CL,AL
 12AA E8CAFF             call CONOUT     ;print it
 12AD 43                 inc BX
 12AE EBF2               jmps pmsg       ;next character and loop

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

                 SELDSK:         ;select disk given by register CL
 12B0 BB0000             mov bx,0000h
 12B3 80F902             cmp cl,2        ;this BIOS only supports 2
 12B6 7318               jnb return      ;return w/ 0000 in BX if ba
 12B8 B080               mov al, 80h
 12BA 80F900             cmp cl,0
 12BD 7502               jne sell        ;drive 1 if not zero
 12BF B040               mov al, 40h     ;else drive is 0
 12C1 A21315     sell:   mov sel_mask,al ;save drive select mask
                                         ;now, we need disk paramete
 12C4 B500               mov ch,0
 12C6 8BD9               mov bx,cx       ;BX = word(CL)
 12C8 B104               mov cl,4
 12CA D3E3               shl bx,cl       ;multiply drive code * 16
                         ;create offset from Disk Parameter Base
 12CC 81C32215           add bx,offset dp_base
                 return:
 12D0 C3                 ret

                 HOME:   ;move selected disk to home position (Track
 12D1 C606161500         mov trk,0       ;set disk i/o to track zero
 12D6 BB1815             mov bx,offset hom_com
 12D9 E83500             call execute
 12DC 74F2               jz return       ;home drive and return if 0
 12DE BB1414             mov bx,offset bad_hom   ;else print
 12E1 E8BEFF             call pmsg       ;"Home Error"
 12E4 EBEB               jmps home       ;and retry

                 SETTRK: ;set track address given by CX
```

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

                  SETSEC: ;set sector number given by cx
 12EB 880E1715            mov sect,cl     ;we only use 8 bits of sect
 12EF C3                  ret

                  SECTRAN: ;translate sector CX using table at [DX]
 12F0 8BD9                mov bx,cx
 12F2 03DA                add bx,dx       ;add sector to tran table a
 12F4 8A1F                mov bl,[bx]     ;get logical sector
 12F6 C3                  ret

                  SETDMA: ;set DMA offset given by CX
 12F7 890E0F15            mov dma_adr,CX
 12FB C3                  ret

                  SETDMAB: ;set DMA segment given by CX
 12FC 890E1115            mov dma_seg,CX
 1300 C3                  ret
                  ;
                  GETSEGT:  ;return address of physical memory table
 1301 BB1D15              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  *
                  ;*                                            *
                  ;**********************************************

                  READ:
 1305 B012                mov al,12h      ;basic read sector command
 1307 EB02                jmps r_w_common

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

                  r_w_common:
 130B BB1415              mov bx,offset io_com ;point to command stri
 130E 884701              mov byte ptr 1[BX],al ;put command into str
                  ;       fall into execute and return

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

 1311 891E0D15            mov last_com,BX ;save command address for r
                  outer_retry:
                          ;allow some retrying
 1315 C6060C150A          mov rtry_cnt,max_retries
```

```
                        retry:
 131A 8B1E0D15                  mov BX,last_com
 131E E88900                    call send_com   ;transmit command to i8271
                        ;       check status poll

 1321 8B1E0D15                  mov BX,last_com
 1325 8A4701                    mov al,1[bx]    ;get command op code
 1328 B90008                    mov cx,0800h    ;mask if it will be "int re
 132B 3C2C                      cmp al,2ch
 132D 720B                      jb exec_poll    ;ok if it is an interrupt t
 132F B98080                    mov cx,8080h    ;else we use "not command b
 1332 240F                      and al,0fh
 1334 3C0C                      cmp al,0ch      ;unless there isn't
 1336 B000                      mov al,0
 1338 7736                      ja exec_exit    ;       any result
                                                ;poll for bits in CH,
                        exec_poll:              ;  toggled with bits in CL

 133A E4A0                      in al,fdc_stat  ;read status
 133C 22C5                      and al,ch
 133E 32C1                      xor al,cl       ;  isolate what we want to
 1340 74F8                      jz exec_poll    ;and loop until it is done

                                                ;Operation complete,
 1342 E4A1                      in al,fdc_rslt  ; see if result code indica
 1344 241E                      and al,1eh
 1346 7428                      jz exec_exit    ;no error, then exit
                                                ;some type of error occurre
 1348 3C10                      cmp al,10h
 134A 7425                      je dr_nrdy      ;was it a not ready drive ?
                                                ;no,
                        dr_rdy: ; then we just retry read or write
 134C FE0E0C15                  dec rtry_cnt
 1350 75C8                      jnz retry       ;  up to 10 times

                        ;       retries do not recover from the
                        ;       hard error

 1352 B400                      mov ah,0
 1354 8BD8                      mov bx,ax       ;make error code 16 bits
 1356 8B9F3B14                  mov bx,errtbl[BX]
 135A E845FF                    call pmsg       ;print appropriate message
 135D E4D8                      in al,cdata     ;flush usart receiver buffe
 135F E82BFF                    call uconecho   ;read upper case console ch
 1362 3C43                      cmp al,'C'
 1364 7425                      je wboot_l      ;cancel
 1366 3C52                      cmp al,'R'
 1368 74AB                      je outer_retry  ;retry 10 more times
 136A 3C49                      cmp al,'I'
 136C 741A                      je z_ret        ;ignore error
 136E 0CFF                      or al,255       ;set code for permanent err
                        exec_exit:
 1370 C3                        ret

                        dr_nrdy:        ;here to wait for drive ready
```

```
1371 E81A00                     call test_ready
1374 75A4                       jnz retry         ;if it's ready now we are d
1376 E81500                     call test_ready
1379 759F                       jnz retry         ;if not ready twice in row,
137B BBAC14                     mov bx,offset nrdymsg
137E E821FF                     call pmsg ;"Drive Not Ready"
                        nrdy01:
1381 E80A00                     call test_ready
1384 74FB                       jz nrdy01         ;now loop until drive ready
1386 EB92                       jmps retry        ;then go retry without decr
                        zret:
1388 2400                       and al,0
138A C3                         ret               ;return with no error code

                        wboot_l:                  ;can't make it w/ a short l
138B E9D3FE                     jmp WBOOT

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

                        test_ready:
138E B640                       mov dh, 40h      ;proper mask if dr 1
1390 F606131580                 test sel_mask,80h
1395 7502                       jnz nrdy2
1397 B604                       mov dh, 04h      ;mask for dr 0 status bit
                        nrdy2:
1399 BB1B15                     mov bx,offset rds_com
139C E80B00                     call send_com
                        dr_poll:
139F E4A0                       in al,fdc_stat   ;get status word
13A1 A880                       test al,80h
13A3 75FA                       jnz dr_poll      ;wait for not command busy
13A5 E4A1                       in al,fdc_rslt   ;get "special result"
13A7 84C6                       test al,dh       ;look at bit for this drive
13A9 C3                         ret              ;return status of ready

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

                        send_com:
13AA E4A0                       in al,fdc_stat
13AC A880                       test al,80h      ;insure command not busy
13AE 75FA                       jnz send_com     ;loop until ready
```

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

 13B0 8A4701                mov al,1[bx]    ;get command byte
 13B3 3C12                  cmp al,12h
 13B5 7504                  jne write_maybe ;if not a read it could be
 13B7 B140                  mov cl,40h
 13B9 EB06                  jmps init_dma   ;is a read command, go set
                    write_maybe:
 13BB 3C0A                  cmp al,0ah
 13BD 7520                  jne dma_exit    ;leave DMA alone if not rea
 13BF B180                  mov cl,80h      ;we have write, not read
                    init_dma:
                    ;we have a read or write operation, setup DMA contr
                    ;       (CL contains proper direction bit)
 13C1 B004                  mov al,04h
 13C3 E6A8                  out dmac_mode,al    ;enable dmac
 13C5 B000                  mov al,00
 13C7 E6A5                  out dmac_cont,al    ;send first byte to con
 13C9 8AC1                  mov al,cl
 13CB E6A5                  out dmac_cont,al   ;load direction register
 13CD A10F15                mov ax,dma_adr
 13D0 E6A4                  out dmac_adr,al    ;send low byte of DMA
 13D2 8AC4                  mov al,ah
 13D4 E6A4                  out dmac_adr,al    ;send high byte
 13D6 A11115                mov ax,dma_seg
 13D9 E6AA                  out fdc_segment,al ;send low byte of segmen
 13DB 8AC4                  mov al,ah
 13DD E6AA                  out fdc_segment,al ;then high segment addre
                    dma_exit:
 13DF 8A0F                  mov cl,[BX]     ;get count
 13E1 43                    inc BX
 13E2 8A07                  mov al,[BX]     ;get command
 13E4 0A061315              or al,sel_mask  ;merge command and drive co
 13E8 E6A0                  out fdc_com,al  ;send command byte
                    parm_loop:
 13EA FEC9                  dec cl
 13EC 7482                  jz exec_exit    ;no (more) parameters, retu
 13EE 43                    inc BX          ;point to (next) parameter
                    parm_poll:
 13EF E4A0                  in al,fdc_stat
 13F1 A820                  test al,20h     ;test "parameter register f
 13F3 75FA                  jnz parm_poll   ;idle until parm reg not fu
 13F5 8A07                  mov al,[BX]
 13F7 E6A1                  out fdc_parm,al ;send next parameter
 13F9 EBEF                  jmps parm_loop  ;go see if there are more p

                    ;*********************************************
                    ;*                                           *
                    ;*               Data Areas                  *
                    ;*                                           *
                    ;*********************************************
  13FB              data_offset     equ offset $

                            dseg
                            org     data_offset     ;contiguous with co
```

```
 13FB 0D0A0D0A      signon  db      cr,lf,cr,lf
 13FF 4D502F4D2D38          db      'MP/M-86 Loader 2.0',cr,lf,0
      36204C6F6164
      657220322E30
      0D0A00

 1414 0D0A486F6D65 bad_hom db      cr,lf,'Home Error',cr,lf,0
      204572726F72
      0D0A00
 1423 0D0A496E7465 int_trp db      cr,lf,'Interrupt Trap Halt',cr,lf,0
      727275707420
      547261702048
      616C740D0A00

 143B 5B145B145B14 errtbl  dw er0,er1,er2,er3
      5B14
 1443 6B147B148814         dw er4,er5,er6,er7
      9914
 144B AC14C014D214         dw er8,er9,erA,erB
      E714
 1453 F7145B145B14         dw erC,erD,erE,erF
      5B14

 145B 0D0A4E756C6C er0     db  cr,lf,'Null Error ??',0
      204572726F72
      203F3F00
   145B            er1     equ er0
   145B            er2     equ er0
   145B            er3     equ er0
 146B 0D0A436C6F63 er4     db  cr,lf,'Clock Error :',0
      6B204572726F
      72203A00
 147B 0D0A4C617465 er5     db  cr,lf,'Late DMA :',0
      20444D41203A
      00
 1488 0D0A49442043 er6     db  cr,lf,'ID CRC Error :',0
      524320457272
      6F72203A00
 1499 0D0A44617461 er7     db  cr,lf,'Data CRC Error :',0
      204352432045
      72726F72203A
      00
 14AC 0D0A44726976 er8     db  cr,lf,'Drive Not Ready :',0
      65204E6F7420
      526561647920
      3A00
 14C0 0D0A57726974 er9     db  cr,lf,'Write Protect :',0
      652050726F74
      656374203A00
 14D2 0D0A54726B20 erA     db  cr,lf,'Trk 00 Not Found :',0
      3030204E6F74
      20466F756E64
      203A00
 14E7 0D0A57726974 erB     db  cr,lf,'Write Fault :',0
```

```
      65204661756C
      74203A00
 14F7 0D0A53656374 erC      db  cr,lf,'Sector Not Found :',0
      6F72204E6F74
      20466F756E64
      203A00
  145B             erD      equ er0
  145B             erE      equ er0
  145B             erF      equ er0
  14AC             nrdymsg  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           io_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       csv0,alv0       ;Check, Alloc Vecto
 1532 51150000     dpe1     dw       xlt1,0000h      ;Translate Table
 1536 00000000              dw       0000h,0000h     ;Scratch Area
 153A 6B154215              dw       dirbuf,dpb1     ;Dir Buff, Parm Blo
 153E 39161A16              dw       csv1,alv1       ;Check, Alloc Vecto
                   ;                 DISKDEF 0,1,26,6,1024,243,64,64,2
                   ;
                   ;        1944:    128 Byte Record Capacity
                   ;         243:    Kilobyte Drive  Capacity
                   ;          64:    32 Byte Directory Entries
                   ;          64:    Checked Directory Entries
                   ;         128:    Records / Extent
                   ;           8:    Records / Block
                   ;          26:    Sectors / Track
                   ;           2:    Reserved  Tracks
                   ;           6:    Sector Skew Factor
                   ;
```

```
  1542             dpb0     equ      offset $          ;Disk Parameter Blo
 1542 1A00                  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 Max
 154B C0                    db       192               ;Alloc0
 154C 00                    db       0                 ;Alloc1
 154D 1000                  dw       16                ;Check Size
 154F 0200                  dw       2                 ;Offset
  1551             xlt0     equ      offset $          ;Translate Table
 1551 01070D13              db       1,7,13,19
 1555 19050B11              db       25,5,11,17
 1559 1703090F              db       23,3,9,15
 155D 1502080E              db       21,2,8,14
 1561 141A060C              db       20,26,6,12
 1565 1218040A              db       18,24,4,10
 1569 1016                  db       16,22
  001F             als0     equ      31                ;Allocation Vector
  0010             css0     equ      16                ;Check Vector Size
                   ;                 DISKDEF 1,0
                   ;
                   ;        Disk 1 is the same as Disk 0
                   ;
  1542             dpb1     equ      dpb0              ;Equivalent Paramet
  001F             als1     equ      als0              ;Same Allocation Ve
  0010             css1     equ      css0              ;Same Checksum Vect
  1551             xlt1     equ      xlt0              ;Same Translate Tab
                   ;                 ENDEF
                   ;
                   ;        Uninitialized Scratch Memory Follows:
                   ;
  156B             begdat   equ      offset $          ;Start of Scratch A
 156B              dirbuf   rs       128               ;Directory Buffer
 15EB              alv0     rs       als0              ;Alloc Vector
 160A              csv0     rs       css0              ;Check Vector
 161A              alv1     rs       als1              ;Alloc Vector
 1639              csv1     rs       css1              ;Check Vector
  1649             enddat   equ      offset $          ;End of Scratch Are
  00DE             datsiz   equ      offset $-begdat ;Size of Scratch Ar
 1649 00                    db       0                 ;Marks End of Modul

 164A              loc_stk rw  32  ;local stack for initialization
  168A             stkbase equ offset $

  168A             lastoff equ offset $
  01A9             tpa_seg equ (lastoff+0400h+15) / 16
  1E57             tpa_len equ 2000h - tpa_seg
 168A 00                   db 0    ;fill last address for GENCMD

                   ;*****************************************************
                   ;*                                                   *
                   ;*         Dummy Data Section                        *
                   ;*                                                   *
```

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

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

APPENDIX 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.A86 file provided with CP/M-86.  This BIOS   *
* allows MP/M-86 operation with the Intel SBC       *
* 86/12, with the SBC 204 controller and with the   *
* SBC 534 expansion interface.  Use this XIOS as    *
* the basis for a customized implementation of      *
* MP/M-86.                                          *
*                                                   *
*****************************************************
*                                                   *
*   (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.)                                 *
*                                                   *
*****************************************************
```

```
        title   '8086 Hardware Interface'

;*****************************************************************
;*
;*                    X  I  O  S  -  8  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  | |
;*               +-----------------+ |
;*               |  XIOS (C and D) |  >   System Data
;*               +-----------------+ |
;*               |  Sysdat         | |
;*               +-----------------+X
;*               |  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  | |
;*               +-----------------+ |
;*               |  XIOS Data      |  >   System Data
;*               +-----------------+ |      Area
;*               |  Sysdat         | |
;*               +-----------------+X
;*               |  XIOS Code      | |
;*               +-----------------+  >   System Code
;*               |  System Code    | |     Modules
;*      low      +-----------------+/
```

```
                        ;*
                        ;*      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)
                        ;*              CX = parameter
                        ;*              DX = second parameter
                        ;*              DS = sysdat (in entry and init)
                        ;*                 = CS elsewhere
                        ;*              ES = User's Data Area
                        ;*
                        ;*      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  0ffffh     ; value of TRUE
=  0000                 false       equ  0          ; value of FALSE
=  0000                 unknown     equ  0          ; value to be filled in
=  0080                 dskrecl     equ  128        ; log. disk record len
=  0020                 fcblen      equ  32         ; size of file control block
=  0008                 pnamsiz     equ  8          ; size of process name
=  0008                 qnamsiz     equ  pnamsiz    ; size of queue name
=  0008                 fnamsiz     equ  pnamsiz    ; size of file name
=  0003                 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
```

```
=  0100            ulen        equ  0100h      ; size of uda
=  0030            pdlen       equ  030h       ; size of Process Descriptor
=  0005            todlen      equ  5          ; size of Time of Day struct
=  0001            flag_tick   equ  1          ; flag 0 = tick flag
=  0002            flag_sec    equ  2          ; flag 1 = second flag
=  0003            flag_min    equ  3          ; flag 2 = minute flag
=  00AA            ldtabsiz    equ  0aah       ; ldtablen=11, 10 entries

                   ;
                   ;       conditional assembly switches
                   ;
   0000            debug            equ     false
                   ;*****MDISK SUPPORT*****
   FFFF            memdisk          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 SBC 534 serial board
                   ;
                           ;por# addres# t# rese# entir# board
   004F            sereset          equ     serbase+0fh
                   ;
                           ;port address to set test mode
   004E            setestmode       equ     serbase+0eh
                   ;
                           ;port address to enable counter/timer
   004C            ctcenable        equ     serbase+0ch
                   ;
                           ;port address to enable uarts
   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
   00B6            ctc2mode         equ     0b6h
   0036            ctc3mode         equ     036h
                   ;
```

```
                                        ;counter/timer load addresses
                                        ; and count values
 0040                ctc0ld          equu    serbase+00h
 0041                ctc1ld          equ     serbase+01h
 0042                ctc2ld          equ     serbase+02h
 0044                ctc3ld          equ     serbase+04h
 0008                cnt0vall        equ     008h
 0000                cnt0valh        equ     000h
 0008                cnt1vall        equ     008h
 0000                cnt1valh        equ     000h
 0008                cnt2vall        equ     008h
 0000                cnt2valh        equ     000h
 0008                cnt3vall        equ     008h
 0000                cnt3valh        equ     000h
                                        ;
                                        ;uart mode and command
                                        ;
 004E                u0mode          equ     04eh
 004E                u1mode          equ     04eh
 004E                u2mode          equ     04eh
 004E                u3mode          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
                     ;

                                        ;
                                        ; console 0
                                        ;
 00D8                c0ioport    equ 0d8h
 00DA                c0stport    equ 0dah
 0002                c0inmsk     equ 02h
 0001                c0outmsk    equ 01h
                                        ;
                                        ; console 1
                                        ;
 0042                c1ioport        equ     serbase+02h
 0043                c1stport        equ     serbase+03h
 0002                c1inmsk         equ     002h
 0001                c1outmsk        equ     001h
                                        ;
                                        ; console 2
                                        ;
 0044                c2ioport        equ     serbase+04h
 0045                c2stport        equ     serbase+05h
 0002                c2inmsk         equ     002h
 0001                c2outmsk        equ     001h
                                        ;
                                        ; console 3
                                        ;
 0046                c3ioport        equ     serbase+06h
 0047                c3stport        equ     serbase+07h
```

```
0002            c3inmsk         equ     002h
0001            c3outmsk        equ     001h
                        ;
                        ; list 0
                        ;
0040            10ioport        equ     serbase+00h
0041            10stport        equ     serbase+01h
0002            10inmsk         equ     002h
0081            10outmsk        equ     081h
                ;
                ;
                ;
                ;*****************************************************
                ;*
                ;*      DISK I/O EQUATES
                ;*
                ;*    Intel iSBC 204 Disk Controller Ports
                ;*
                ;*****************************************************

00A0            base204         equ 0a0h        ;SBC204 assigned addr

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

000A            max_retries     equ 10          ;retries on disk i/o
                                                ;before perm error
                ;*****MDISK SUPPORT*****
2000            mdiskbase       equ 2000h       ;base address of mdisk
                ;**********************

                ;*****************************************************
                ;*
                ;*      SUP/RTM EQUATES
                ;*
                ;*****************************************************

0100            tracebit        equ     0100H

008E            f_dispatch  equ 142     ; MPM dispatch func #
008F            f_terminate equ 143     ; MPM terminate func #
0083            f_polldev   equ 131     ; MPM polldevice func #
0085            f_flagset   equ 133     ; MPM flagset func #
```

```
    0006          p_flag   equ     word ptr 06H     ; PD flag field
    0008          p_name   equ     byte ptr 08H     ; PD Name field
    0020          p_cns    equ     byte ptr 020H    ; PD console field
    0002          pf_keep equ      02H              ; KEEP bit in p_flag

                          ;flag assignments

    0001          tick_flag   equ 1
    0002          sec_flag    equ 2

                          ;device # assignments for POLL DEV

    0000          c0indev      equ 00h      ;console 0 input device
    0001          c1indev      equ 01h      ;console 1 input device
    0002          c2indev      equ 02h      ;console 2 input device
    0003          c3indev      equ 03h      ;console 3 input device
    0004          c0outdev     equ 04h      ;console 0 output device
    0005          c1outdev     equ 05h      ;console 1 output device
    0006          c2outdev     equ 06h      ;console 2 output device
    0007          c3outdev     equ 07h      ;console 3 output device
    0008          l0outdev     equ 08h      ;list 0 output device
    0009          flpy_poll_dev equ 09h     ;floppy disk poll device

                  ;
                  ;       system data area must preceed code
                  ;       area for 8080 model of the XIOS
                  ;
=                 include sysdat.lib
=
=                 ;*****************************************************
=                 ;*
=                 ;*       System Data Area
=                 ;*
=                 ;*****************************************************
=
=                         org     00h
=0000             supmod          rw      2
=
=                         org     038h
=  0038           dispatcher      equ     (offset $)
=
=                         org     040h
=0040             mpmseg          rw      1
=
=                         org     044h
=0044             endseg          rw      1
=
=                         org     068h
=0068             rlr             rw      1        ;Ready List Root
=
=                         org     072h
=0072             thrdrt          rw      1        ;Process Thread Root
=0074             qlr             rw      1        ;Queue List Root
=
```

```
=                           org     078h
=0078               version         rw      1       ;addr. version in SUP
=007A               vernum          rw      1       ;MPM-86 w/BDOS v3.0
=007C               mpmvernum       rw      1       ;MPM-86 Version 1.0
=007E               tod             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                 jmp init        ;system initialization
1003 E9E101                 jmp entry       ;xios entry point

1006 0000           sysdat          dw      0       ;Sysdat Segment
  1008              supervisor      equ     offset $
1008                                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                   pushf ! cli
100E FF362E18               push stoppoll
1012 C7062E18FFFF           mov stoppoll,true
1018 1E2E8E1E0610           push ds ! mov ds,sysdat ;DS = system data area
101E 2E8B366800             mov si,rlr              ;SI -> current pd
1023 8A5420                 mov dl,p_cns[si]        ;DL = def console #
1026 1F                     pop ds
                    ploop:
1027 8A07                   mov al,[bx]             ; get next char
1029 3C00740C               cmp al,0 ! jz pmsg_ret  ; return if zero
102D 8AC8                       mov cl,al           ; CL = character
```

```
102F 5253                      push dx ! push bx    ; save console,posit.
1031 E8FC01                    call conout          ; print it
1034 5B5A                      pop bx ! pop dx      ; restore posit.,cons.
1036 43EBEE                    inc bx ! jmps ploop  ; inc and loop
                    pmsg_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 0000           ax_save         dw      0
1045 0000           zero            dw      0

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

                                    ;save context
1047 1E2E8E1E0610          push ds ! mov ds,sysdat
104D 2E8C163F10            mov tickint_ss,ss
1052 2E89264110            mov tickint_sp,sp
1057 2EA34310              mov ax_save,ax
105B 8CC8                  mov ax,cs
105D 8ED0                  mov ss,ax
105F BC6715                mov sp,offset tickint_tos
1062 2EFF364310            push ax_save
1067 535152                push bx ! push cx ! push dx
106A 55565706              push bp ! push si ! push di ! push es
106E 8ED8                  mov ds,ax

                                    ; check to set second flag

1070 FE0E3118750D          dec tick_count ! jnz do_tick_flag
1076 C60631183C                mov tick_count,60
107B BA0200                    mov dx,sec_flag
107E B185E87601                mov cl,f_flagset ! call supif
                    do_tick_flag:
                                    ; check to set tick flag

1083 803E3018FF75          cmp clockon,true ! jne tick_done
     08
108A BA0100                    mov dx,tick_flag
108D B185E86701                mov cl,f_flagset ! call supif

                    tick_done:      ;restore context
```

```
1092 075F5E5D               pop es ! pop di ! pop si ! pop bp
1096 5A595B58               pop dx ! pop cx ! pop bx ! pop ax
109A 2E8E163F10             mov ss,tickint_ss
109F 2E8B264110             mov sp,tickint_sp
                                    ;force dispatch
10A4 1F                     pop ds
                            ; jmp intdisp

                    ;=======
                    intdisp:
                    ;=======

10A5 2EFF2E3800             jmpf cs:dword ptr .dispatcher

                    ;========
                    int_trap:       ;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 8CC8                   mov ax,cs
10AC 2E8E1E0610             mov ds,sysdat

                                    ; print first 6 chars of PD Name
10B1 2E8B1E6800             mov bx,rlr
10B6 83C308                 add bx,p_name
10B9 C647063A               mov byte ptr 6[bx],':'
10BD C6470700               mov byte ptr 7[bx],0
10C1 E848FF                 call pmsg

                                    ; print Illegal Interrupt message
10C4 BB0C15                 mov bx,offset int_trp
10C7 E842FF                 call pmsg

                                    ; terminate process
10CA 2E8B1E6800             mov bx,rlr
10CF 816706FDFF             and p_flag[bx],not pf_keep
10D4 B98F00                 mov cx,f_terminate
10D7 BAFFFF                 mov dx,0ffffh
10DA CDE0                   int 224
10DC F4                     hlt     ;hard stop
                                    ;the terminate returned !!!!

                    ;*****************************************************
                    ;*
                    ;*      INITIALIZATION CODE AREA
                    ;*
                    ;*      Inter-Module Interface Routines
                    ;*
```

```
                        ;*****************************************************

                        ;====
                        init:           ; XIOS system initialization routine.
                        ;====
                        ;       The INIT routine initializes all necessary
                        ;       hardware
                        ;
                        ; -called from SUP init. routine with CALLF
                        ;
                        ; -Interrupt 224 is reinitialized by SUP later
                        ; -It is okay to turn on interrupts at any time
                        ;       a STI is performed immediately after RETF
                        ;
                        ; -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=Sysdat
                        ; -If assembled with separate Code and Data,
                        ;       CS=Code (ORGed at 0) DS=Sysdat
                        ;
                        ; -This example shows 8080 model
                        ;
                        ;       input:  DS = sysdat segment address
                        ;
                        ;-----------------------------------------------------
                        ;       SYSTEM INITIALIZATION AREA
                        ;-----------------------------------------------------
                                        ;Save sysdat seg addr in case we need
                                        ;to see system data area.  Set DS=CS

10DD 1E                         push ds                 ;save DS on stack for
                                                        ;exit

                                        ;initialize segment registers

10DE 2E8C1E0610                 mov sysdat,ds           ; save sysdat for
                                                        ; sysdat access

                                        ;place copy of SUPMOD in data segment
                                        ;into Code Segment (supervisor)

10E3 BB0000                     mov bx,offset supmod
10E6 BE0810                     mov si,supervisor
10E9 8B072E898400               mov ax,[bx] ! mov cs:[si],ax
     00
10F0 8B47022E8984               mov ax,2[bx] ! mov cs:2[si],ax
     0200

                                        ;Make copy of Interrupt Routines
                                        ;access point to dispatcher in
                                        ;Code Segment

10F8 0E1F                       push cs ! pop ds        ; DS = CS
```

```
 10FA FC                      cld                ;set forward direction

                                        ;stack switch since we are doing
                                        ; i/o with polled devices when
                                        ; printing login message
                                        ; interrupts are known to be off
                                        ; here so no need to save flags,
                                        ; disable, and restore flags

 10FB 8C166715                mov initss,ss
 10FF 89266915                mov initsp,sp
 1103 8CC88ED0                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
 110B 06                      push es            ;ES must be saved
 110C B80000                  mov ax,0
 110F 8ED8                    mov ds,ax
 1111 8EC0                    mov es,ax          ;set ES and DS to zero

                                        ;setup interrupt 0 to trap routine

 1113 C7060000AA10            mov .0,offset int_trap
 1119 8C0E0200                mov .2,CS
 111D BF0400                  mov di,4
 1120 BE0000                  mov si,0           ;then propagate
 1123 B9FE01                  mov cx,510         ;trap vector to
 1126 F3A5                    rep movsw          ;int 255

 1128 07                      pop es
 1129 1F                      pop ds             ;restore DS,ES

                              endif

                         ;-------------------------------------------------------
                         ;       CHARACTER I/O INITIALIZATION
                         ;-------------------------------------------------------
                         ;
                         ;
                              ;**********************************************
                              ;*
                              ;*    sbc 534 serial board initialization
                              ;*
                              ;**********************************************
                         ;
 112A B000                    mov al,0
 112C E64F                    out sereset,al          ;reset serial board
 112E E64E                    out setestmode,al       ;set test mode to off
 1130 E64C                    out ctcenable,al        ;enable ctc addressing
```

```
                          ;
                                          ;start clock for port 0
 1132 B036E643                    mov al,ctc0mode ! out ctc0to2md,al
 1136 B008E640                    mov al,cnt0vall ! out ctc0ld,al
 113A B000E640                    mov al,cnt0valh ! out ctc0ld,al
                          ;
                                          ;start clock for port 1
 113E B076E643                    mov al,ctc1mode ! out ctc0to2md,al
 1142 B008E641                    mov al,cnt1vall ! out ctc1ld,al
 1146 B000E641                    mov al,cnt1valh ! out ctc1ld,al
                          ;
                                          ;start clock for port 2
 114A B0B6E643                    mov al,ctc2mode ! out ctc0to2md,al
 114E B008E642                    mov al,cnt2vall ! out ctc2ld,al
 1152 B000E642                    mov al,cnt2valh ! out ctc2ld,al
                          ;
                                          ;start clock for port 3
 1156 B036E647                    mov al,ctc3mode ! out ctc3to5md,al
 115A B008E644                    mov al,cnt3vall ! out ctc3ld,al
 115E B000E644                    mov al,cnt3valh ! out ctc3ld,al
                          ;
 1162 E64D                        out uartenable,al       ;enable uart addressing
                          ;
                                          ;initialize port 0
 1164 B04EE641                    mov al,u0mode ! out l0stport,al
 1168 B037E641                    mov al,u0cmd ! out l0stport,al
                          ;
                                          ;initialize port 1
 116C B04EE643                    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 068EC1                      push es ! mov es,cx
 118A BF0000B8E5E5                mov di,0 ! mov ax,0e5e5h
 1190 2639057405                  cmp es:[di],ax ! je mdisk_end
 1195 B90020                          mov cx,2000h
 1198 F3AB                            rep stos ax
                          mdisk_end:
 119A 07                          pop es
```

```
                                ;
                                endif
                                ;*************************

                        ;---------------------------------------------------------
                        ;       SUP/RTM INITIALIZATION
                        ;---------------------------------------------------------
                                        ;Initialize Clock Tick

                                if not debug
                                                ;set up tick interrupt vector
                                                ;tick causes interrupt 22
119B 2BC01E8ED8                 sub ax,ax ! push ds ! mov ds,ax
11A0 C70688004710               mov word ptr .088h,offset tick_int
11A6 8C0E8A00                   mov word ptr .08ah,cs
11AA 1F                         pop ds
                                                ;setup ticks to occur every
                                                ; 1/60th of a second
11AB C60631183C                 mov tick_count,60
11B0 B034E6D6                   mov al,034h ! out 0d6h,al  ;PIT ch.0=mode 2
                                                ;set # of 1/1.2288e6 seconds
                                                ;20480=5000h=1/60th second
11B4 B000E6D0                   mov al,000h ! out 0d0h,al  ;low count
11B8 B050E6D0                   mov al,050h ! out 0d0h,al  ;high count
                                                ;set up interrupt controller
11BC B013E6C0                   mov al,013h ! out 0c0h,al  ;ICW1
11C0 B020E6C2                   mov al,020h ! out 0c2h,al  ;ICW2
                                                           ;  = base interrupt
11C4 B00FE6C2                   mov al,00fh ! out 0c2h,al  ;ICW4,
                                                           ; auto EOI,
                                                           ; 8086 mode
11C8 B0FBE6C2                   mov al,0fbh ! out 0c2h,al  ;OCW2,
                                                           ; interrupt mask,
                                                           ; only 2
                                endif
                        ;---------------------------------------------------------
                        ;       INITIALIZATION EXIT
                        ;---------------------------------------------------------
                                        ;allow poll_device mechanism to work

11CC C7062E180000               mov stoppoll,false

                                        ;print optional message on Console 0

11D2 BBEE14                     mov bx,offset signon
11D5 E834FE                     call pmsg

                                        ;restore stack
                                        ; all stack switches must be in
                                        ; critical areas (interrupts off).

11D8 9C58                       pushf ! pop ax
11DA FA                         cli
11DB 8E166715                   mov ss,initss
11DF 8B266915                   mov sp,initsp
```

```
 11E3 50                  push ax
 11E4 9D                  popf

                                  ;return back to BDOS
 11E5 1F                  pop ds
 11E6 CB                  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

 11E7 FC                  cld                         ;clear D flag
 11E8 8CCB8EDB            mov bx,cs ! mov ds,bx       ; (only 8080 model)
 11EC B400D1E0            mov ah,0 ! shl ax,1         ;call routine
 11F0 8BD8FF97A814        mov bx,ax ! call functab[bx]
 11F6 8BD8                mov bx,ax                   ;BX=AX
 11F8 CB                  retf                        ;All Done

                    ;=====          ======================
                    supif:          ; Supervisor Interface
                    ;=====          ======================
                    ;
                    ;       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

 11F9 B500                mov ch,0
 11FB 2EFF1E0810C3        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 B500D1E1               mov ch,0 ! shl cx,1
1205 8BD9                   mov bx,cx
1207 8B97AC15               mov dx,consttbl[bx]
120B 8ADE                   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:
1218 C3                     ret

                    ;=====
                    conin:      ; Function 1:  Console Input
                    ;=====
                    ;       input:  CL = console device number
                    ;       output: AL = character

1219 B500D1E1               mov ch,0 ! shl cx,1
121D 8BD9                   mov bx,cx
121F 8B97BC15               mov dx,conintbl[bx]
1223 8ADE                   mov bl,dh       ;BL = poll device no.
1225 B600                   mov dh,0        ;DX = i/o port address

                    ; input routine for console device

1227 B70052                 mov bh,0 ! push dx
122A E81C025A               call rtm_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
                    ;               DL = console device #
                    ;       output: None

1230 B600D1E2               mov dh,0 ! shl dx,1
1234 8BDA                   mov bx,dx
1236 8B97C415               mov dx,conouttbl[bx]
```

```
 123A 8ADE                    mov bl,dh        ;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 B700E80402              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 B600D1E2                mov dh,0 ! shl dx,1
 124D 8BDA                    mov bx,dx
 124F 8B97CE15                mov dx,louttbl[bx]
 1253 8ADE                    mov bl,dh        ;BL = poll device no.
 1255 B600                    mov dh,0         ;DX = list port address

                        ; output routine for list device

 1257 5152                    push cx ! push dx
 1259 B700E8EB01              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
                        ;               under MP/M-86
                        ;       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 B500D1E1                mov ch,0 ! shl cx,1
 1269 8BD9                    mov bx,cx
 126B 8B97CC15                mov dx,loutsttbl[bx]
 126F 8ADE                    mov bl,dh        ;BL = status mask
 1271 B600                    mov dh,0         ;DX = output port address

                        ;       find output status of List device
```

```
1273 EC                         in al,dx
1274 22C3                       and al,bl
1276 3AC3                       cmp al,bl
1278 B0FF                       mov al,0ffh
127A 7402                       jz gstat
127C B000                           mov al,0
127E C3                 gstat:  ret

                        ;==========
                        maxconsole: ; Function 20:  Maximum Consoles
                        ;==========
                        ;       input:  None
                        ;       output: AL = number of consoles

127F B80400C3                   mov ax,nconsoles ! ret

                        ;=======
                        maxlist:    ; Function 21:  Maximum List Devices
                        ;=======
                        ;       input:  None
                        ;       output: AL = number of consoles

1283 B80100C3                   mov ax,nlists ! ret

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

                        ;====
                        HOME:       ; Function 6:  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

1287 C606AB1600                 mov trk,0       ;set disk i/o to track zero
128C 33DB                       xor bx,bx
128E 8A1EB416                   mov bl,disk     ;index into disk home routine
1292 D1E3                       shl bx,1
1294 FFA7C516                   jmp dskhomtbl[bx]

                        flpy_home:
1298 BBAD16                     mov bx,offset hom_com
129B E8A800                     call execute    ;home drive
                        hom_ret:
129E C3                         ret             ; and return

                        ;======
                        SELDSK:     ; Function 7:  Select Disk
                        ;======
```

```
                         ;       input:  CL = disk to be selected
                         ;       output: AX = 0 if illegal disk
                         ;                  = 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 8BF1                        mov si,cx          ;to physical disk device
12A7 8A84B516                    mov al,dtrntbl[si]
12AB 3CFF                        cmp al,0ffh        ;valid disk select?
12AD 740C                        jz sel_ret         ;if not valid return
                                                    ;else compute the
                                                    ; disk parameter address
12AF A2B416                          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 8BC3                        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
                                                    ; track address
12C2 C3                          ret

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

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

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

12C8 890EA416                    mov dma_adr,CX
12CC C3                          ret

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

```
                         ;*   Read and Write entry points transfer one *
                         ;*   sector of 128 bytes to/from the current  *
                         ;*   DMA address using the current disk drive *
                         ;*                                            *
                         ;*********************************************

                         ;====
                         READ:        ; Function 11:  Read
                         ;====
                         ;        input:  None
                         ;        output: AL = 00h if no error occured
                         ;                   = 01h if error occured

12CD 33DB                         xor bx,bx
12CF 8A1EB416                     mov bl,disk      ;index into disk read routin
12D3 D1E3                         shl bx,1
12D5 FFA7CB16                     jmp dskrdtbl[bx]

                                  ;***** MDISK SUPPORT *****
                                  if memdisk
                                  ;
                         mdsk_read:
12D9 E81E00                           call mdisk_calc
12DC 8B3EA416BE00                     mov di,dma_adr ! mov si,0
     00
12E3 068E06A616                       push es ! mov es,dma_seg
12E8 1E8ED8                           push ds ! mov ds,ax
12EB B94000F3A5                       mov cx,64 ! rep movs ax,ax
12F0 1F07                             pop ds ! pop es
12F2 B80000C3                         mov ax,0 ! ret
                                  ;
                                  endif
                                  ;*************************

                         flpy_read:
12F6 B012                         mov al,12h      ;basic read sector command
12F8 EB46                         jmps r_w_common

                                  ;***** MDISK SUPPORT *****
                                  if memdisk
                                  ;
                         mdisk_calc:
12FA B7008A1EAB16                 mov bh,0 ! mov bl,trk
1300 B81A00F7E3                   mov ax,26 ! mul bx
1305 B7008A1EAC16                 mov bh,0 ! mov bl,sect
130B 03C3B103                     add ax,bx ! mov cl,3
130F D3E0050120                   shl ax,cl ! add ax,mdiskbase+1
1314 C3                           ret
                                  ;
                                  endif
                                  ;*************************

                         ;=====
                         WRITE:       ; Function 12:  Write
                         ;=====
```

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

1315 33DB                        xor bx,bx
1317 8A1EB416                    mov bl,disk      ;index into disk write routine
131B D1E3                        shl bx,1
131D FFA7D116                    jmp dskwrttbl[bx]

                                 ;***** MDISK SUPPORT *****
                                 if memdisk
                                 ;
                         mdsk_write:
1321 E8D6FF                          call mdisk_calc
1324 BF00008B36A4                    mov di,0 ! mov si,dma_adr
     16
132B 068EC0                          push es ! mov es,ax
132E 1E8E1EA616                      push ds ! mov ds,dma_seg
1333 B94000F3A5                      mov cx,64 ! rep movs ax,ax
1338 1F07                            pop ds ! pop es
133A B80000C3                        mov ax,0 ! ret
                                 ;
                                 endif
                                 ;*************************

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

                         r_w_common:
1340 BBA916                      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,80h
1348 F606B41601                  test disk,1      ;A drive is even phys. drive
134D 7502                        jnz execl        ;B drive is odd phys. drive
134F B040                            mov al,40h
1351 A2A816              execl:  mov sel_mask,al
1354 891EA216                    mov last_com,BX ;save command address
                                                 ;for retries
                         outer_retry:
                                 ;allow some retrying
1358 C606A1160A                  mov rtry_cnt,max_retries
                         retry:
135D 8B1EA216                    mov BX,last_com
1361 E86700                      call send_com    ;transmit command to i8271
```

```
                        ;       check status poll

 1364 8B1EA216                  mov BX,last_com
 1368 8A4701                    mov al,1[bx]      ;get command op code
 136B B90008                    mov cx,0800h      ;mask if it will be "int re
 136E 3C2C                      cmp al,2ch
 1370 720B                      jb flpy_poll      ;ok if it is an interrupt t
 1372 B98080                        mov cx,8080h  ;else use "not command bus
 1375 240F                          and al,0fh
 1377 3C0C                          cmp al,0ch    ; unless there isn't
 1379 B000                          mov al,0
 137B 7731                          ja exec_exit  ; any result
                                                  ;poll for bits in CH,
                        flpy_poll:                ;  toggled with bits in CL
 137D 890EB216                  mov status_mask,cx
 1381 BB0900E8C200              mov bx,flpy_poll_dev ! call rtm_poll
                                                  ;Operation complete,
 1387 E4A1                      in al,fdc_rslt    ; see if result indicates
 1389 241E                      and al,1eh        ; an error
 138B 7421                      jz exec_exit      ;no error, then exit
                                                  ;some type of error occurred
 138D 3C10                      cmp al,10h
 138F 7508                      jne dr_rdy        ;was it a not ready drive ?
                                    ;yes, here to wait for drive ready
 1391 E81B00                        call test_ready
 1394 7503                          jnz dr_rdy  ;if ready try again
 1396 E81600                            call test_ready
                                        ;if not ready twice in row,
                        dr_rdy: ; then we just retry read or write
 1399 FE0EA116                  dec rtry_cnt
 139D 75BE                      jnz retry         ; up to 10 times

                        ;       retries do not recover from the
                        ;       hard error

 139F 241E                      and al,1eh        ;setup error table index
 13A1 B400                      mov ah,0
 13A3 8BD8                      mov bx,ax         ;make error code 16 bits
 13A5 8B9FD015                  mov bx,errtbl[BX]
 13A9 E860FC                    call pmsg         ;print appropriate message
 13AC 0CFF                      or al,255         ;set code for permanent error
                        exec_exit:
 13AE C3                        ret

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

                        test_ready:
 13AF B640                      mov dh, 40h       ;proper mask if dr 1
 13B1 F606A81680                test sel_mask,80h
```

```
13B6 7502               jnz nrdy2
13B8 B604                   mov dh, 04h ;mask for dr 0 status bit
                  nrdy2:
13BA BBB016             mov bx,offset rds_com
13BD E80B00             call send_com
                  dr_poll:
13C0 E4A0                   in al,fdc_stat  ;get status word
13C2 A880                   test al,80h
13C4 75FA                   jnz dr_poll ;wait for not command busy
13C6 E4A1               in al,fdc_rslt  ;get "special result"
13C8 84C6               test al,dh      ;look at bit for this drive
13CA C3                 ret             ;return status of ready

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

                  send_com:
13CB E4A0                   in al,fdc_stat
13CD A880                   test al,80h      ;insure command not busy
13CF 75FA                   jnz send_com     ;loop until ready

                        ;check to initialize for a DMA operation

13D1 8A4701             mov al,1[bx]    ;get command byte
13D4 3C12               cmp al,12h
13D6 7504               jne write_maybe ;if not read it maybe write
13D8 B140                   mov cl,40h  ;is a read command, go set DMA
13DA EB06                   jmps init_dma
                  write_maybe:
13DC 3C0A               cmp al,0ah
13DE 7520               jne dma_exit    ;leave DMA if not read/write
13E0 B180                   mov cl,80h  ;we have write, not read
                  init_dma:
                  ; read or write operation, setup DMA controller
                  ;       (CL contains proper direction bit)
13E2 B004               mov al,04h
13E4 E6A8               out dmac_mode,al     ;enable dmac
13E6 B000               mov al,00
13E8 E6A5               out dmac_cont,al     ;1st byte to ctrl port
13EA 8AC1               mov al,cl
13EC E6A5               out dmac_cont,al     ;load direction register
13EE A1A416             mov ax,dma_adr
13F1 E6A4               out dmac_adr,al      ;send low byte of DMA
13F3 8AC4               mov al,ah
13F5 E6A4               out dmac_adr,al      ;send high byte
13F7 A1A616             mov ax,dma_seg
13FA E6AA               out fdc_segment,al   ;send low byte of seg addr
13FC 8AC4               mov al,ah
13FE E6AA               out fdc_segment,al   ;then high segment address
```

```
                        dma_exit:
1400 8A0F                       mov cl,[BX]      ;get count
1402 43                         inc BX
1403 8A07                       mov al,[BX]      ;get command
1405 0A06A816                   or al,sel_mask   ;merge command and drive code
1409 E6A0                       out fdc_com,al   ;send command byte
                        parm_loop:
140B FEC9                       dec cl
140D 749F                       jz 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 EBEF                       jmps parm_loop  ;see if more parameters

                        flpy_poll_stat:
                        ;--------------
                        ; See if current operation complete

141C 2E8B0EB216                 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 flpy_poll_nrdy
142B B0FF                           mov al,0ffh
                        flpy_poll_nrdy:
142D C3                         ret

                        ;=======
                        SECTRAN:     ; 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 83FA007503                 cmp dx,0 ! jne sectranl
1433 8BC1C3                         mov ax,cx ! ret
                        sectranl:
1436 8BD9                       mov bx,cx
1438 03DA                       add bx,dx        ;add sector to tran table addr
143A 8A1F                       mov bl,[bx]      ;get logical sector
143C 32FF                       xor bh,bh
143E 8BC3                       mov ax,bx
1440 C3                         ret

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

```
                        ;        input:  CX = Segment of DMA buffer
                        ;        output: None

 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
                        ;*
                        ;*****************************************************
                        ;--------
                        rtm_poll:        ;check dev, if not ready, do rtm poll
                        ;--------
                        ; Check poll condition.
                        ; If not ready,Call MPM Poll Device Routine
                        ;        input:  BX = device #

 1449 538BCB                     push bx ! mov cx,bx
 144C E814005B                   call polldev ! pop bx
 1450 3CFF740E                   cmp al,0ffh ! je pllret
 1454 833E2E18FF74                   cmp stoppoll,true ! je rtm_poll
      EE
                        ;---------
                        rtm_poll1:       ; do poll_dev with no pretest
                        ;---------
 145B 8BD3B183                       mov dx,bx ! mov cl,f_polldev
 145F E997FD                         jmp supif
 1462 C3                pllret: ret

                        ;=======
                        polldev:    ; Function 17:  Poll Device
                        ;=======
                        ;        input:  CL = device number
                        ;        output: AL = 000h if not ready
                        ;                   = 0ffh if ready

 1463 B7008AD9                   mov bh,0 ! mov bl,cl
 1467 D1E3                       shl bx,1
 1469 FFA7DA14                   jmp polltbl[bx]

                        ;=======
                        strtclk:    ; Function 18:  Start Clock
                        ;=======
                        ; Enable Flagsets on Tick Interrupts
```

```
                      ;       input:  None
                      ;       output: None

146D C6063018FF               mov clockon,true
1472 C3                       ret

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

1473 C606301800               mov clockon,false
1478 C3                       ret

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

1479 B8FFFFC3                 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 B18E                     mov cl,f_dispatch
1480 E876FD                   call supif
1483 E9F8FF                   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.
```

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

1486 909090909090                  nop ! nop ! nop ! nop ! nop ! nop
148C 909090909090                  nop ! nop ! nop ! nop ! nop ! nop
1492 909090909090                  nop ! nop ! nop ! nop ! nop ! nop
1498 909090909090                  nop ! nop ! nop ! nop ! nop ! nop
149E 909090909090                  nop ! nop ! nop ! nop ! nop ! nop
14A4 9090                          nop ! nop

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

14A6                       endcode rw      0

                                   DSEG
                                   org     (offset endcode + 1) and 0fffeh
                                           ;org to an even word offset
                                           ;current UDA for MPM calls
14A6                       udaseg          rw      1

                           ;-----------------------------------------------
                           ;       XIOS FUNCTION TABLE
                           ;-----------------------------------------------
14A8 0112                  functab dw      const           ; 0-console status
14AA 1912                          dw      conin           ; 1-console input
14AC 3012                          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      SELDSK          ; 7-select disk
14B8 BE12                          dw      SETTRK          ; 8-set track
14BA C312                          dw      SETSEC          ; 9-set sector
14BC C812                          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      sectran         ;14-sector translate
14C6 4114                          dw      setdmab         ;15-set DMA base
14C8 7914                          dw      getsegt         ;16-get segment table
14CA 6314                          dw      polldev         ;17-poll device
14CC 6D14                          dw      strtclk         ;18-start clock
14CE 7314                          dw      stopclk         ;19-stop clock
14D0 7F12                          dw      maxconsole      ;20-maximum consoles
14D2 8312                          dw      maxlist         ;21-max list devices
14D4 7D14                          dw      selmemory       ;22-select memory
14D6 7E14                          dw      idle            ;23-idle
14D8 4614                          dw      flushbuf        ;24-flush buffer
```

```
                     ;-----------------------------------------------------
                                 ;
                                 ; 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 0 out
 14E4 0112                       dw     const             ;05-con 1 out
 14E6 0112                       dw     const             ;06-con 2 out
 14E8 0112                       dw     const             ;07-con 3 out
 14EA 6512                       dw     listst            ;08-list 0 out
 14EC 1C14                       dw     flpy_poll_stat    ;09-dsk status

 14EE 0D0A           signon  db         cr,lf
 14F0 4D502F4D2D38           db         'MP/M-86 V2.0 for SBC 8612',cr,lf,0
362056322E3020666F
      722053424320
      383631320D0A
      00

 150C 20556E696E69 int_trp db           ' Uninitialized Interrupt',cr,lf,0
      7469616C697A
      656420496E74
      657272757074
      0D0A00

 1527                loc_stk rw  32  ;local stack for initialization
   1567              stkbase equ offset $
 1567                tickint_tos     rw     0

 1567                initss          rw     1
 1569                initsp          rw     1
 156B                                rw     32
 15AB                initstack       rw     0


                     ;*****************************************************
                     ;*
                     ;*       CHARACTER I/O DATA AREA
                     ;*
                     ;*****************************************************
                                 ;
                                 org    ((offset $) + 1) and 0fffeh
                                 ;
                                 ; console i/o table for
                                 ; status mask and port address
                                 ;
 15AC DA02           consttbl    dw     (c0inmsk shl 8) or c0stport
 15AE 4302                       dw     (c1inmsk shl 8) or c1stport
 15B0 4502                       dw     (c2inmsk shl 8) or c2stport
 15B2 4702                       dw     (c3inmsk shl 8) or c3stport
 15B4 DA01                       dw     (c0outmsk shl 8) or c0stport
```

```
15B6 4301                         dw      (c1outmsk shl 8) or c1stport
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 c0ioport
15BE 4201                         dw      (c1indev shl 8) or c1ioport
15C0 4402                         dw      (c2indev shl 8) or c2ioport
15C2 4603                         dw      (c3indev shl 8) or c3ioport
                                  ;
                                  ; console output table for
                                  ; poll device no. and port address
                                  ;
15C4 D804          conouttbl      dw      (c0outdev shl 8) or c0ioport
15C6 4205                         dw      (c1outdev shl 8) or c1ioport
15C8 4406                         dw      (c2outdev shl 8) or c2ioport
15CA 4607                         dw      (c3outdev shl 8) or c3ioport
                                  ;
                                  ; list i/o table for
                                  ; status mask and port address
                                  ;
15CC 4181          loutsttbl      dw      (l0outmsk shl 8) or l0stport
                                  ;
                                  ; list output for
                                  ; poll device no. and port address
                                  ;
15CE 4008          louttbl        dw      (l0outdev shl 8) or l0ioport

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

15D0 F015F015F015 errtbl   dw er0,er1,er2,er3
     F015
15D8 001610161D16          dw er4,er5,er6,er7
     2E16
15E0 411655166716          dw er8,er9,erA,erB
     7C16
15E8 8C16F015F015          dw erC,erD,erE,erF
     F015

15F0 0D0A4E756C6C 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
```

```
      20444D41203A
      00
161D 0D0A49442043 er6      db  cr,lf,'ID CRC Error :',0
      524320457272
      6F72203A00
162E 0D0A44617461 er7      db  cr,lf,'Data CRC Error :',0
      204352432045
      72726F72203A
      00
1641 0D0A44726976 er8      db  cr,lf,'Drive Not Ready :',0
      65204E6F7420
      526561647920
3A00
1655 0D0A57726974 er9      db  cr,lf,'Write Protect :',0
      652050726F74
      656374203A00
1667 0D0A54726B20 erA      db  cr,lf,'Trk 00 Not Found :',0
      3030204E6F74
      20466F756E64
      203A00
167C 0D0A57726974 erB      db  cr,lf,'Write Fault :',0
      65204661756C
      74203A00
168C 0D0A53656374 erC      db  cr,lf,'Sector Not Found :',0
      6F72204E6F74
      20466F756E64
      203A00
  15F0                erD      equ er0
  15F0                erE      equ er0
  15F0                erF      equ er0
  1641                nrdymsg equ er8

16A1 00               rtry_cnt db 0   ;disk error retry counter
16A2 0000             last_com dw 0   ;address of last command string
16A4 0000             dma_adr  dw 0   ;dma offset stored here
16A6 0000             dma_seg  dw 0   ;dma segment stored here
16A8 40               sel_mask db 40h ;select mask, 40h or 80h

                      ;        Various command strings for i8271

16A9 03               io_com  db 3    ;length
16AA 00               rd_wr   db 0    ;read/write function code
16AB 00               trk     db 0    ;track #
16AC 00               sect    db 0    ;sector #

16AD 022900           hom_com db 2,29h,0       ;home drive command
16B0 012C             rds_com db 1,2ch         ;read status command

16B2 0000             status_mask     dw      0        ;mask for flpy_poll


16B4 00               disk            db      0        ;physical disk selected

                               ;       logical to physical disk
                               ;       translation table
```

```
                                ;          A    B    C    D
 16B5 0001FFFF      dtrntbl db          000h,001h,0ffh,0ffh
                                ;          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 memdisk
 16C1 02FFFFFF              db          002h,0ffh,0ffh,0ffh
                                endif
                                ;***********************
                                if not memdisk
                                db          0ffh,0ffh,0ffh,0ffh
                                endif

                                ;           disk home routine table
 16C5 9812          dskhomtbl           dw      flpy_home        ;A drive
 16C7 9812                              dw      flpy_home        ;B drive
                    ;*****MDISK SUPPORT*****
                    if memdisk
 16C9 9E12                              dw      hom_ret          ;M drive
                    endif
                    ;***********************

                                ;           disk read routine table
 16CB F612          dskrdtbl            dw      flpy_read        ;A drive
 16CD F612                              dw      flpy_read        ;B drive
                    ;*****MDISK SUPPORT*****
                    if memdisk
 16CF D912                              dw      mdsk_read        ;M drive
                    endif
                    ;***********************

                                ;           disk write routine table
 16D1 3E13          dskwrttbl           dw      flpy_write       ;A drive
 16D3 3E13                              dw      flpy_write       ;B drive
                    ;*****MDISK SUPPORT*****
                    if memdisk
 16D5 2113                              dw      mdsk_write       ;M drive
                    endif
                    ;***********************

=                               include singles.lib ;read in disk definitions
=                   ;           DISKS 3
=  16D7             dpbase  equ $                    ;Base of Disk Parameter
=16D7 16170000      dpe0    dw  xlt0,0000h           ;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      dpe1    dw  xlt1,0000h           ;Translate Table
=16EB 00000000              dw  0000h,0000h          ;Scratch Area
=16EF 3F170717              dw  dirbuf,dpb1          ;Dir Buff, Parm Block
=16F3 0D18EE17              dw  csv1,alv1            ;Check, Alloc Vectors
=16F7 00000000      dpe2    dw  xlt2,0000h           ;Translate Table
```

```
=16FB 00000000               dw      0000h,0000h     ;Scratch Area
=16FF 3F173017               dw      dirbuf,dpb2     ;Dir Buff, Parm Block
=1703 2D181D18               dw      csv2,alv2       ;Check, Alloc Vectors
=                    ;               DISKDEF 0,1,26,6,1024,243,64,64,2
=                    ;
=                    ;         1944: 128 Byte Record Capacity
=                    ;          243: Kilobyte Drive  Capacity
=                    ;           64: 32 Byte Directory Entries
=                    ;           64: Checked Directory Entries
=                    ;          128: Records / Extent
=                    ;            8: Records / Block
=                    ;           26: Sectors / Track
=                    ;            2: Reserved  Tracks
=                    ;            6: Sector Skew Factor
=                    ;
=  1707              dpb0    equ     offset $        ;Disk Parameter Block
=1707 1A00                   dw      26              ;Sectors Per Track
=1709 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 01070D13               db      1,7,13,19
=171A 19050B11               db      25,5,11,17
=171E 1703090F               db      23,3,9,15
=1722 1502080E               db      21,2,8,14
=1726 141A060C               db      20,26,6,12
=172A 1218040A               db      18,24,4,10
=172E 1016                   db      16,22
=  001F              als0    equ     31              ;Allocation Vector Size
=  0010              css0    equ     16              ;Check Vector Size
=                    ;               DISKDEF 1,0
=                    ;
=                    ;       Disk 1 is the same as Disk 0
=                    ;
=  1707              dpb1    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
=                    ;               DISKDEF 2,1,26,,1024,127,64,0,0
=                    ;
=                    ;         1016: 128 Byte Record Capacity
=                    ;          127: Kilobyte Drive  Capacity
=                    ;           64: 32 Byte Directory Entries
=                    ;            0: Checked Directory Entries
=                    ;          128: Records / Extent
=                    ;            8: Records / Block
=                    ;           26: Sectors / Track
=                    ;            0: Reserved  Tracks
=                    ;
```

```
=  1730            dpb2      equ     offset $          ;Disk Parameter Block
=1730 1A00                   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 C0                     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
=                  ;                 ENDEF
=                  ;
=                  ;         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
=17EE              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            datsiz    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
                                                       ;for internal printing
                                                       ;of strings when true

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

 1831 3C           tick_count        db      60
                   ;
                   ;         scratch area
                   ;
 1832 CCCCCCCCCCCC                   dw      0cccch,0cccch,0cccch
 1838 CCCCCCCCCCCC                   dw      0cccch,0cccch,0cccch
 183E CCCCCCCCCCCC                   dw      0cccch,0cccch,0cccch
 1844 CCCCCCCCCCCC                   dw      0cccch,0cccch,0cccch
 184A CCCCCCCCCCCC                   dw      0cccch,0cccch,0cccch
 1850 CCCCCCCCCCCC                   dw      0cccch,0cccch,0cccch
 1856 CCCCCCCCCCCC                   dw      0cccch,0cccch,0cccch
```

```
185C 00                   db 0    ;fill last address for GENCMD
```

APPENDIX D

Blocking/Deblocking Algorithms

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

```
wrall   equ     0                   ;write to allocated
wrdir   equ     1                   ;write to directory
wrual   equ     2                   ;write to unallocated
;
;*****************************************************
;*                                                   *
;*      The BIOS entry points given below show the   *
;*      code which is relevant to deblocking only.   *
;*                                                   *
;*****************************************************
seldsk:
        ;select disk
        ;is this the first activation of the drive?
        test DL,1                   ;lsb = 0?
        jnz selset
        ;this is the first activation, clear host buff
        mov hstact,0
        mov unacnt,0
selset:
        mov al,cl ! cbw             ;put in AX
        mov sekdsk,al               ;seek disk number
        mov cl,4 ! shl al,cl        ;times 16
        add ax,offset dpbase
        mov bx,ax
        ret
;
home:
        ;home the selected disk
        mov al,hstwrt               ;check for pending write
        test al,al
        jnz homed
        mov hstact,0                ;clear host active flag
homed:
        mov cx,0                    ;now, set track zero
;       (continue HOME routine)
        ret
;
settrk:
        ;set track given by registers CX
        mov sektrk,CX               ;track to seek
        ret
;
setsec:
        ;set sector given by register cl
        mov seksec,cl               ;sector to seek
        ret
;
setdma:
        ;set dma address given by CX
        mov dma_off,CX
        ret
;
setdmab:
        ;set segment address given by CX
        mov dma_seg,CX
```

```
          ret
;
sectran:
          ;translate sector number CX with table at [DX]
          test DX,DX      ;test for hard skewed
          jz notran       ;(blocked must be hard skewed)
          mov BX,CX
          add BX,DX
          mov BL,[BX]
          ret
no_tran:
          ;hard skewed disk, physical = logical sector
          mov BX,CX
          ret
;
read:
          ;read the selected CP/M sector
          mov unacnt,0              ;clear unallocated counter
          mov readop,1              ;read operation
          mov rsflag,1              ;must read data
          mov wrtype,wrual          ;treat as unalloc
          jmp rwoper                ;to perform the read
;
write:
          ;write the selected CP/M sector
          mov readop,0              ;write operation
          mov wrtype,cl
          cmp cl,wrual              ;write unallocated?
          jnz chkuna                ;check for unalloc
;
;         write to unallocated, set parameters
;
          mov unacnt,(blksiz/128) ;next unalloc recs
          mov al,sekdsk             ;disk to seek
          mov unadsk,al             ;unadsk = sekdsk
          mov ax,sektrk
          mov unatrk,ax             ;unatrk = sektrk
          mov al,seksec
          mov unasec,al             ;unasec = seksec
;
chkuna:
          ;check for write to unallocated sector
;
          mov bx,offset unacnt      ;point "UNA" at UNACNT
          mov al,una ! test al,al ;any unalloc remain?
          jz alloc                  ;skip if not
;
;         more unallocated records remain
          dec al                    ;unacnt = unacnt-1
          mov una,al
          mov al,sekdsk             ;same disk?
          mov BX,offset unadsk
          cmp al,una                ;sekdsk = unadsk?
          jnz alloc                 ;skip if not
;
```

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

```
        cmp al,hstdsk           ;sekdsk = hstdsk?
        jnz nomatch
;
;       same disk, same track?
        mov ax,hsttrk
        cmp ax,sektrk           ;host track same as seek
        jnz nomatch
;
;       same disk, same track, same buffer?
        mov al,sekhst
        cmp al,hstsec           ;sekhst = hstsec?
        jz match                ;skip if match
nomatch:
        ;proper disk, but not correct sector
        mov al, hstwrt
        test al,al              ;"dirty" buffer ?
        jz filhst               ;no, don't need to write
        call writehst           ;yes, clear host buff
;       (check errors here)
;
filhst:
        ;may have to fill the host buffer
        mov al,sekdsk ! mov hstdsk,al
        mov ax,sektrk ! mov hsttrk,ax
        mov al,sekhst ! mov hstsec,al
        mov al,rsflag
        test al,al              ;need to read?
        jz filhstl
;
        call readhst            ;yes, if 1
;       (check errors here)
;
filhstl:
        mov hstwrt,0            ;no pending write
;
match:
        ;copy data to or from buffer depending on "readop"
        mov al,seksec           ;mask buffer number
        and ax,secmsk           ;least signif bits are masked
        mov cl, 7 ! shl ax,cl   ;shift left 7 (* 128 = 2**7)
;
;       ax has relative host buffer offset
;
        add ax,offset hstbuf    ;ax has buffer address
        mov si,ax               ;put in source index register
        mov di,dma_off          ;user buffer is dest if readop
;
        push DS ! push ES       ;save segment registers
;
        mov ES,dma_seg          ;set destseg to the users seg
                                ;SI/DI and DS/ES is swapped
                                ;if write op
        mov cx,128/2            ;length of move in words
        mov al,readop
        test al,al              ;which way?
```

```
        jnz     rwmove          ;skip if read
;
;       write operation, mark and switch direction
        mov hstwrt,1            ;hstwrt = 1 (dirty buffer now)
        xchg si,di              ;source/dest index swap
        mov ax,DS
        mov ES,ax
        mov DS,dma_seg          ;setup DS,ES for write
;
rwmove:
        cld ! rep movs AX,AX    ;move as 16 bit words
        pop ES ! pop DS         ;restore segment registers
;
;       data has been moved to/from host buffer
        cmp wrtype,wrdir        ;write type to directory?
        mov al,erflag           ;in case of errors
        jnz return_rw           ;no further processing
;
;       clear host buffer for directory write
        test al,al              ;errors?
        jnz return_rw           ;skip if so
        mov hstwrt,0            ;buffer written
        call writehst
        mov al,erflag
return_rw:
        ret
;
;*****************************************************
;*                                                   *
;* WRITEHST performs the physical write to the host  *
;* disk, while READHST reads the physical disk.      *
;*                                                   *
;*****************************************************
writehst:
        ret
;
readhst:
        ret
;
;*****************************************************
;*                                                   *
;* Use the GENDEF utility to create disk def tables  *
;*                                                   *
;*****************************************************
dpbase  equ     offset $
;       disk parameter tables go here
;
;*****************************************************
;*                                                   *
;* Uninitialized RAM areas follow, including the     *
;* areas created by the GENDEF utility listed above. *
;*                                                   *
;*****************************************************
sek_dsk rb      1               ;seek disk number
sek_trk rw      1               ;seek track number
```

```
sek_sec rb      1               ;seek sector number
;
hst_dsk rb      1               ;host disk number
hst_trk rw      1               ;host track number
hst_sec rb      1               ;host sector number
;
sek_hst rb      1               ;seek shr secshf
hst_act rb      1               ;host active flag
hst_wrt rb      1               ;host written flag
;
una_cnt rb      1               ;unalloc rec cnt
una_dsk rb      1               ;last unalloc disk
una_trk rw      1               ;last unalloc track
una_sec rb      1               ;last unalloc sector
;
erflag  rb      1               ;error reporting
rsflag  rb      1               ;read sector flag
readop  rb      1               ;1 if read operation
wrtype  rb      1               ;write operation type
dma_seg rw      1               ;last dma segment
dma_off rw      1               ;last dma offset
hstbuf  rb      hstsiz          ;host buffer
        end
```

INDEX

MP/M-86™ Operating System

SYSTEM GUIDE

Corrections to the First Printing - September 1981

Compiled October 5, 1981

PAGE 43

```
XIOS Function 12: WRITE

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

PAGE 46

```
XIOS Function 24: FLUSH BUFFERS

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

PAGE 80

The example generation of MPMLDR, although correct in itself, does not generate an MPMLDR loading at paragragh 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. ...

to:

    byte location 04000H. ...

PAGE 83

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

PAGE 109

APPENDIX C - EXAMPLE XIOS LISTING

change:

    ... XIOS.A86 file provided with CP/M-86. ...

to:

    ... 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.

PAGE 143

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