MP/M 1.0

A Multi-Programming Monitor Control Program
for
Microcomputer System Development

FUNCTIONAL SPECIFICATION

9 August 1979

Table of Contents

1.Ø PRODUCT IDENTIFICATION
1.1 Overview..1

2.Ø PRODUCT RATIONALE
2.1 Design Objectives.................................2

3.Ø SYSTEM REQUIREMENTS
3.1 Hardware Environment..............................3

4.Ø PERFORMANCE OBJECTIVES
4.1 Memory Size.......................................4
4.2 Speed...4
4.3 Reliability.......................................4
4.4 Maintainability...................................4

5.Ø DOCUMENTATION
5.1 External and End User.............................5
5.2 Internal and OEM..................................5

6.Ø DESCRIPTION OF FEATURES
6.1 Description of Basic I/O Facilities...............6
6.1.1 Cold Start......................................6
6.1.2 Warm Start......................................6
6.1.3 Console Status..................................6
6.1.4 Console Character In............................6
6.1.5 Console Character Out...........................6
6.1.6 List Character Out..............................6
6.1.7 Punch Character Out.............................7
6.1.8 Reader Character In.............................7
6.1.9 Move Head to Home Position......................7
6.1.1Ø Select Disk....................................7
6.1.11 Set Track Number...............................7
6.1.12 Set Sector Number..............................7
6.1.13 Set DMA Address................................7
6.1.14 Read Disk......................................7
6.1.15 Write Disk.....................................7
6.1.16 List Device Status.............................7
6.1.17 Sector Translate...............................7
6.2 Description of Extended I/O Facilities............8
6.2.1 Memory Selection / Protection...................8
6.2.2 Device Polling..................................8
6.2.3 Start Clock.....................................9
6.2.4 Stop Clock......................................9
6.2.5 Exit Region.....................................1Ø
6.2.6 Maximum Console.................................1Ø
6.2.7 System Initialization...........................11
6.2.8 Interrupt Service Routines......................11
6.2.9 Time Base Management............................13
6.2.1Ø BIOS External Jump Vector......................14
6.3 Description of Basic Disk Operating System........15
6.3.1 System Reset....................................15

6.3.2 Read Console....................................15
6.3.3 Write Console...................................15
6.3.4 Read Reader.....................................15
6.3.5 Write Punch.....................................15
6.3.6 Write List......................................15
6.3.7 Direct Console I/O..............................15
6.3.8 Get I/O Status..................................15
6.3.9 Set I/O Status..................................15
6.3.10 Print Buffer...................................15
6.3.11 Read Buffer....................................15
6.3.12 Interrogate Console Ready......................15
6.3.13 Return Version Number..........................15
6.3.14 Reset Disk System..............................15
6.3.15 Select Disk....................................15
6.3.16 Open File......................................16
6.3.17 Close File.....................................16
6.3.18 Search First...................................16
6.3.19 Search Next....................................16
6.3.20 Delete File....................................16
6.3.21 Read Record....................................16
6.3.22 Write Record...................................16
6.3.23 Make File......................................16
6.3.24 Rename File....................................16
6.3.25 Interrogate Login..............................16
6.3.26 Interrogate Disk...............................16
6.3.27 Set DMA Address................................16
6.3.28 Interrogate Allocation.........................16
6.3.29 Write Protect Assigned Disk....................16
6.3.30 Interrogate R/O Bit Vector.....................16
6.3.31 Set File Attributes............................16
6.3.32 Get Address of Disk Params.....................16
6.3.33 Set/Get User Code..............................16
6.3.34 Read Random....................................16
6.3.35 Write Random...................................16
6.3.36 Compute File Size..............................16
6.3.37 Set Random Record..............................16

6.4 Description of Extended Disk Operating System.......17
6.4.1 Absolute Memory Request.........................17
6.4.2 Relocatable Memory Request......................18
6.4.3 Memory Free.....................................18
6.4.4 Poll Device.....................................18
6.4.5 Flag Wait.......................................19
6.4.6 Flag Set..19
6.4.7 Make Queue......................................20
6.4.8 Open Queue......................................22
6.4.9 Delete Queue....................................23
6.4.10 Read Queue.....................................23
6.4.11 Conditional Read Queue.........................24
6.4.12 Write Queue....................................24
6.4.13 Conditional Write Queue........................25
6.4.14 Delay..25
6.4.15 Dispatch.......................................26
6.4.16 Terminate Process..............................26
6.4.17 Create Process.................................27
6.4.18 Set Priority...................................29

6.4.19 Attach Console....................................29
6.4.20 Detach Console....................................29
6.4.21 Set Console.......................................30
6.4.22 Assign Console....................................30
6.4.23 Send CLI Command..................................31
6.4.24 Call Resident System Procedure....................31
6.4.25 Parse Filename....................................32
6.4.26 Get Console Number................................32
6.4.27 System Data Address...............................32
6.4.28 Get Date and Time.................................33
6.5 Console Commands / Operator Interface..................34
6.5.1 Run Program..34
6.5.2 Abort Program......................................34
6.5.3 Run Resident System Process........................34
6.5.4 Detach Program.....................................35
6.5.5 Attach Program.....................................35
6.5.6 Line Editing and Output Control....................35
6.6 Commonly Used System Programs.........................37
6.6.1 Get/Set User Code..................................37
6.6.2 Erase File...37
6.6.3 Type File..37
6.6.4 File Directory.....................................37
6.6.4 Rename File..37
6.6.6 Text Editor..37
6.6.7 Peripheral Interchange Program.....................37
6.6.8 Assembler..37
6.6.9 Submit...38
6.6.10 Status..38
6.6.11 Dump..38
6.6.12 HexCom (Load).....................................38
6.6.13 Concat..38
6.6.14 Genmod..38
6.6.15 Dynamic Debugging Tool (DDT)......................38
6.7 Extended System Programs..............................39
6.7.1 System Status......................................39
6.7.2 Spooler..39
6.7.3 Time and Date......................................40
6.7.4 Scheduler..40
7.0 SYSTEM GENERATION AND LOADING
7.1 System File Components................................41
7.1.1 System Data..41
7.1.2 Customized BIOS....................................41
7.1.3 Nucleus..42
7.1.4 Resident System Processes..........................42
7.2 Gensys..42
7.3 Loader..43

Appendix

A. Flag Assignments....................................44

B. Priority Assignments..................................45

C. Sample Basic I/O System...............................46

D. Sample Page Relocatable Program.......................69

E. Sample Resident System Process........................75

1.0 PRODUCT IDENTIFICATION

Name: Multi-Programming monitor control program for microcomputer system development

Mnemonic: MP/M 1.0

1.1 Overview

The purpose of a multi-programming monitor control program is to provide a microcomputer system development tool which enables multiple users to develop and debug software using a single microcomputer.

2.0 PRODUCT RATIONALE

2.1 Design Objectives

The MP/M 1.0 operating system is intended to be an upward compatible version of CP/M 2.0 with a number of added facilities. These added facilities are contained in new logical sections of MP/M called the extended I/O system (XIOS) and the extended disk operating system (XDOS). As an upward compatible version, users will be able to easily make the transition from CP/M 2.0 to the MP/M 1.0 operating system. In fact, existing CP/M 2.0 *.COM files can be run under MP/M 1.0, providing that the program has been correctly written. That is, only BDOS calls are made for I/O, no direct BIOS calls are allowed. There must also be at least 4 bytes of extra stack in the CP/M 2.0 *.COM program.

The following basic facilities are provided:

a. Multi-terminal support
b. Multi-Programming at a single terminal
c. Concurrency of I/O and CPU operations
d. Inter-process communication, mutual exclusion and synchronization
e. Ability to operate in sequential, polled or interrupt driven environments
f. System timing functions
g. Logical interrupt system utilizing flags
h. Selection of system options at system generation time
i. Dynamic system configuration at load time

The following optional facilities are provided:

a. Spooling list files to the printer
b. Scheduling programs to be run by date and time
c. Displaying complete system run-time status
d. Setting and reading of the date and time

3.0 SYSTEM REQUIREMENTS

3.1 Hardware Environment

The hardware environment for MP/M 1.0 must include an 8080 or Z80 CPU, a minimum of 32K of memory, one or more consoles, 1 to 16 floppy disk drives, and a clock/timer interrupt.

The distributed form of the MP/M 1.0 operating system is configured for a polled I/O environment with a single console. Multiple processes can be run in this mode. To improve the system performance and capability the following incremental hardware additions can be utilized by the operating system:

a. Full Interrupt System
b. Banked Memory
c. Multiple Consoles

4.Ø PERFORMANCE OBJECTIVES

4.1 Memory Size

The basic MP/M 1.Ø operating system should require no more than 16K bytes of memory when configured for a single console. Each additional console will require 256 bytes.

Optional resident system processes can be specified at system generation which will require varying amounts of memory.

4.2 Speed

When MP/M 1.Ø is configured for a single console and is executing a single process, its speed will approximate that of CP/M 2.Ø. In environments where either multiple processes and/or users are running the speed of each individual process will be degraded in proportion to the amount of I/O and compute resources required. A process which performs a large amount of I/O in proportion to computing will exhibit only minor speed degradation. This also applies to a process that performs a large amount of computing, but is running concurrently with other processes that are largely I/O bound. On the other hand, significant speed degradation will occur in environments in which more than one compute bound process is running.

4.3 Reliability

Reliability of the file structure is enhanced by storage allocation methods which ensure that, in the event of a catastrophic hardware failure such as power fail or program error, the integrity of the file system is maintained.

At the user interface, parameters to critical system calls are checked to determine if the integrity of the resident operating system would be adversely affected by the requested function.

4.4 Maintainability

The MP/M 1.Ø operating system is designed so that it can be independently maintained by OEMs. This places a significant requirement on the maintainability of this software product. Both the data structures utilized by the operating system and the algorithms implemented to manipulate the data structures are described in detail in the design specification. The coding practices employed, and in particular the use of a high level language, should facilitate the maintainence.

5.0 DOCUMENTATION

5.1 External and End User

A form of this MP/M 1.0 Functional Specification is used to generate a User's Guide and an Alteration Guide. This level of documentation describes how to use and generate the operating system. There are no data structures or implementation algorithms presented. However, there are numerous examples illustrating each of the operating system primitives.

5.2 Internal and OEM

The internal and OEM documentation is provided in the MP/M 1.0 Design Specification. This document describes the data structures and implementation algorithms of the operating system. It is intended that this level of documentation will enable an OEM to maintain his own system. Included with the internal and OEM documentation are relevant portions of the source listings for the operating system. The use of a high level implementation language simplifies the documentation task. The listings, used in conjunction with the Design Specification, describe the purpose of each procedure, entry and exit conditions, the data structures manipulated, and the details of the algorithms.

6.Ø DESCRIPTION OF FEATURES

The intent of this section is to provide a detailed description of MP/M 1.Ø features. Emphasis has been placed on describing each of the system primitives. This is in contrast to an MP/M 1.Ø User's Guide, prepared using this functional specificaton, which would emphasize the operator interface to the system. The operator interface is covered in this document in section 6.5 on console commands / operator interface.

Because MP/M 1.Ø is a multi-programming system its primitives are considerably more complex than those of a sequential operating system such as CP/M 2.Ø. For this reason many cf the operating system primitives may be reserved for priviledged execution modes. Primitives in this class include those of memory allocation, process creation and termination.

6.1 Description of Basic I/O Facilities

In general, the Basic I/O System (BIOS) facilities are identical to that of CP/M 2.Ø. Therefore, the reader is referred to the Digital Research document titled "CP/M System Alteration Guide" to obtain a description of the BIOS operations. Only exceptions to CP/M 2.Ø BIOS are noted here.

6.1.1 Cold Start

The BIOS cold start procedure can be used for any device initialization left undone by the bootstrap. Typically the cold start procedure simply jumps to perform a BDOS system reset.

6.1.2 Warm Start

The BIOS warm start procedure makes a BDOS system reset call which terminates the calling process.

6.1.3 Console Status

The BIOS console status procedure is identical to CP/M 2.Ø with the exception that the console number for polling is passed to the procedure in the D register.

6.1.4 Console Character In

The BIOS console character in procedure is identical to CP/M 2.Ø with the exception that the console number for input is passed to the procedure in the D register.

6.1.5 Console Character Out

The BIOS console character out procedure is identical to CP/M 2.Ø with the exception that the console number for output is passed to the procedure in the D register.

6.1.6 List Character Out

6.1.7 Punch Character Out

The BIOS punch character out procedure is not supported. BDOS calls to write punch are defaulted to write console.

6.1.8 Reader Character In

The BIOS reader character in procedure is not supported. BDOS calls to read reader are defaulted to read console.

6.1.9 Move Head to Home Position

6.1.10 Select Disk

6.1.11 Set Track Number

6.1.12 Set Sector Number

6.1.13 Set DMA Address

6.1.14 Read Disk

6.1.15 Write Disk

6.1.16 List Device Status

6.1.17 Sector Translate

6.2 Description of Extended I/O Facilities

The extended I/O facilities include the hardware environment dependent code to poll devices, handle interrupts and perform memory management functions.

6.2.1 Memory Selection/Protection

Each time a process is dispatched to run a call is made to the XIOS memory protection procedure. If the hardware environment has memory bank selection/protection it can use the passed parameter to select/protect areas of memory. The passed parameter is a pointer to a memory descriptor from which the memory base, size, attributes and bank of the executing process can be determined. Thus, all other regions of memory can to be write protected.

```
 SELMEMORY:
         ...                ; BC -> MEMORY DESCRIPTOR
         ...                ;         BASE BYTE,
         ...                ;         SIZE BYTE,
         ...                ;         ATTRIB BYTE,
         ...                ;         BANK BYTE;

         RET
```

6.2.2 Device Polling Routines

In hardware environments where there are no interrupts a polled environment can be created by coding an XIOS device poll handler. The device poll handler (POLLDEVICE) is called by the XDOS with the device to be polled in the C register as a single parameter. The user written POLLDEVICE procedure can be coded to access the device polling routines via a table which contains the addresses of the device polling procedures. An association is made between a device number to be polled and the polling procedure itself. The polling procedures must return a value of ØFFH in the accumulator if the device is ready, or ØØH if the device is not ready.

```
 POLLDEVICE:
         MVI     B,Ø
         LXI     H,DEVPTB
         DAD     B
         DAD     B
         MOV     A,M
         INX     H
         MOV     H,M
         MOV     L,A
         PCHL

 DEVPTB:                            ; DEVICE POLLING TABLE
         DW      CON1IN
```

```
        DW      CON2IN
        ...
        DW      CON1OUT
        DW      CON2OUT
        ...

CON1IN:                         ; CONSOLE 1 INPUT POLL
        IN      CNS1
        ANI     RXRDY
        RZ                      ; RETURNS ZERO FOR NOT READY
        MVI     A,ØFFH
        RET                     ; RETURNS FFH FOR READY

CON2IN:
        IN      CNS2
        ANI     RXRDY
        RZ
        MVI     A,ØFFH
        RET

        ...
```

6.2.3 Start Clock

When a process delays for a specified number of ticks of the system time unit, the start clock procedure is called.

The purpose of the STARTCLOCK procedure is to eliminate unneccessary system clock interrupt overhead when there are not any delayed processes.

In some hardware environments it is not acutally possible to shut off the system time unit clock while still maintaining the one second flag used for the purposes of keeping time of day. In this situation the STARTCLOCK procedure simply sets a boolean variable to true, indicating that there is a delayed process. The clock interrupt handler can then determine if system time unit flag is to be set by testing the boolean.

```
STARTCLOCK:
        MVI     A,CLKSTRT
        OUT     CLK1PORT
        RET


        -OR-


STARTCLOCK:
        MVI     A,ØFFH
        STA     TICKING
        RET
```

6.2.4 Stop Clock

When the system delay list is emptied the stop clock procedure is called.

The purpose of the STOPCLOCK procedure is to eliminate unneccessary system clock interrupt overhead when there are no delayed processes.

In some hardware environments it is not acutally possible to shut off the system time unit clock while still maintaining the one second flag used for the purposes of keeping time of day. (i.e. a single clock/timer interrupt source is used.) In this situation the STOPCLOCK procedure simply sets a boolean variable to false, indicating that there are no delayed processes. The clock interrupt handler can then determine if the system time unit flag is to be set by testing the boolean.

```
STOPCLOCK:
        MVI     A,CLKSTP
        OUT     CLK1PORT
        RET


        -OR-


STOPCLOCK:
        XRA     A
        STA     TICKING
        RET
```

6.2.5 Exit Region

The purpose of the exit region procedure is to test a preempted flag, set by the interrupt handler, enabling interrupts if preempted is false. This procedure allows interrupt service routines to make MP/M system calls, leaving interrupts disabled until completion of the interrupt handling.

```
EXITREGION:
        LDA     PREEMPTED
        ORA     A
        RNZ
        EI
        RET
```

6.2.6 Maximum Console

The purpose of the maximum console procedure is to enable the calling program to determine the number of physical consoles which the BIOS is capable of supporting. The number of physical consoles is returned in the A register.

```
MAXCONSOLE:
```

```
          MVI     A,NMBCNS
          RET
```

6.2.7 System Initialization

The purpose of the system initialization procedure is to perform required MP/M cold start initialization. Typical initialization includes setting up interrupt jump vectors, interrupt masks, and initializing the disk file system.

```
 DISKINIT         EQU     13

 SYSINIT:
          MVI     A,0C3H  ; STORE JUMP @ RESTART 7
          STA     0038H
          LXI     H,INTHNDLR
          SHLD    0039H

          LDA     INTMSK  ; INITIALIZE INTERRUPT MASK
          OUT     IMSKPORT

          MVI     C,DISKINIT
          CALL    XDOS    ; INTIALIZE DISK FILE SYSTEM

          ...
          RET
```

6.2.8 Interrupt Service Routines

The MP/M 1.0 operating system is designed to work with virtually any interrupt architecture, be it flat or vectored. The function of the code operating at the interrupt level is to save the required registers, determine the cause of the interrupt, and to set an appropriate flag. Operation of the flags are described in sections 6.4.5 and 6.4.6. Briefly, flags are used to synchronize asynchronous processes. One process, such as an interrupt service routine, sets a particular flag while another process waits for the flag to be set.

At a logical level above the physical interrupts the flags can be regarded as providing 256 levels of virtual interrupts. Thus, logical interrupt handlers wait on flags to be set by the physical interrupt handlers. This mechanism allows a common XDOS to operate on all microcomputers, regardless of the hardware environment.

As an example consider a hardware environment with a flat interrupt structure. That is, a single interrupt level is provided and devices must be polled to determine the cause of the interrupt. Once the interrupt cause is determined a specific flag is set indicating that that particular interrupt has occurred.

At the conclusion of the interrupt processing a jump should be made to the MP/M dispatcher. This is done by jumping to the PDISP entry point. The effect of this jump is to give the processor to the highest priority ready process, usually the process readied by setting the flag in the interrupt handler, and then to enable interrupts before jumping to resume execution of the process.

```
FLAGSET EQU     134

INTRPT:                             ; INTERRUPT ENTRY
        PUSH    PSW
        PUSH    B
        PUSH    D
        PUSH    H

        MVI     A,0FFH
        STA     PREEMPT             ; SET PREEMPTED TRUE

        IN      CNSIN1              ; POLL CONSOLE 1
        ANI     RXRDY
        JZ      POLL01              ; JUMP TO POLL NEXT
        MVI     E,CIN1
        JMP     FOUNDINTR
POLL01:
        IN      CNSIN2              ; POLL CONSOLE 2
        ANI     RXRDY
        JZ      POLL02              ; JUMP TO POLL NEXT
        MVI     E,CIN2
        JMP     FOUNDINTR
POLL02:
        ...

FOUNDINTR:
        MVI     C,FLAGSET
        CALL    XDOS                ; CALL XDOS TO SET FLAG

        XRA     A
        STA     PREEMPT             ; SET PREEMPTED FALSE

        POP     H
        POP     D
        POP     B
        POP     PSW
        JMP     PDISP               ; JUMP TO DISPATCHER
```

Note in the previous example that the interrupted processes stack is used to save the registers and to make the XDOS call. If this is not acceptable, that is the user process has insufficient stack, a separate stack must be used.

The technique of using a local stack in the interrupt handler is shown below:

```
INTHND:
        PUSH    PSW             ; SAVE A & FLAGS
        SHLD    SVDHL           ; SAVE HL
        LXI     H.0
        DAD     SP
        SHLD    SVDSP           ; SAVE USERS STK PTR
        LXI     SP,INTSTK+48    ; USE LOCAL INTRPT STK
        PUSH    D
        PUSH    B

        ...                     ; NORMAL INTERRUPT HANDLING

        ...

        POP     B
        POP     D
        LHLD    SVDSP
        SPHL                    ; RESTORE USERS STK PTR
        LHLD    SVDHL           ; RESTORE HL
        POP     PSW             ; RESTORE A & FLAGS
        JMP     PDISP           ; JUMP TO DISPATCHER

        ...

SVDHL:  DS      2
SVDSP:  DS      2
INTSTK: DS      48
```

6.2.9 Time Base Management

The time base management provided by the BIOS performs the operations of setting the system tick and one second flags. As described in sections 6.2.3 and 6.2.4 the start and stop clock procedures control the system tick operation. The one second flag operation is logically separate from the system tick operation even though it may physically share the same clock/timer interrupt source.

The purpose of the system time unit tick procedure is to set flag #1 at system time unit intervals. The system time unit is used by MP/M to manage the delay list.

The purpose of the one second flag procedure is to set flag #2 at each second of real time. Flag #2 is used by MP/M to maintain a time of day clock.

The following example illustrates the handling of a single clock/timer interrupt which provides continuous interrupts each 16.67 milliseconds.

```
CLK60HZ:
                                ; 60 HZ CLOCK INTERRUPT ENTRY
        LDA     TICKING
        ORA     A               ; TEST TICKING, TRUE INDICATES
                                ;  SYSTEM TIME UNIT TICK REQD
        JZ      NOTICKING
        MVI     C,FLAGSET
        MVI     E,1
        CALL    XDOS                ; SET FLAG #1
NOTICKING:
        LXI     H,CNT60
        DCR     M                   ; DECREMENT 60 TICK COUNTER
        JNZ     NOT1SEC             ; JUMP IF NOT ONE SECOND
        MVI     M,60                ; RESET COUNTER
        MVI     C,FLAGSET
        MVI     E,2
        CALL    XDOS                ; SET FLAG #2
NOT1SEC:
        ...                         ; CONTINUE OTHER PROCESSING

        ...

CNT60:  DB      60                  ; 60 HZ COUNTER
TICKING:
        DS      1                   ; BOOLEAN SET ON/OFF BY CLOCK
                                    ;  START/STOP PROCEDURES
```

6.2.10 BIOS External Jump Vector

In order for the BIOS to access the BDOS a jump vector is dynamically built by the MP/M loader and placed directly below the base address of the BIOS. The jump vector contains three entry points which provide access to the MP/M dispatcher, XDOS and BDOS.

The following code illustrates the equates used to access the jump table:

```
BASE    EQU     0000H   ; BASE OF THE BIOS

PDISP   EQU     BASE-3  ; MP/M DISPATCHER
XDOS    EQU     PDISP-3 ; MP/M BDOS/XDOS

...
CALL    XDOS            ; CALL TO XDOS THRU JUMP VECTOR
...
```

6.3 Description of Basic Disk Operating System

In general, the Basic Disk Operating System (BDOS) facilities are identical to that of CP/M 2.Ø. Therefore, the reader is referred to the Digital Research document titled "CP/M Interface Guide" to obtain a description of the BDOS operations. Only exceptions to CP/M 2.Ø BDOS are noted here.

6.3.1 System Reset

When a user program performs the BDOS system reset operation the user program process is terminated, usually returning control to the terminal message processor.

6.3.2 Read Console

6.3.3 Write Console

6.3.4 Read Reader

The BDOS read reader is not supported in MP/M 1.Ø. BDOS read reader calls are defaulted to read console operations.

6.3.5 Write Punch

The BDOS write punch is not supported in MP/M 1.Ø. BDOS write punch calls are defaulted to write console operations.

6.3.6 Write List

6.3.7 Direct Console I/O

6.3.8 Get I/O Status

The BDOS get I/O status is not supported in MP/M 1.Ø. BDOS get I/O status calls are treated as a no operation.

6.3.9 Set I/O Status

The BDOS set I/O status is not supported in MP/M 1.Ø. BDOS set I/O status calls are treated as a no operation.

6.3.1Ø Print Buffer

6.3.11 Read Buffer

6.3.12 Interrogate Console Ready

6.3.13 Return Version Number

6.3.14 Reset Disk System

The BDOS reset disk system call is qualified in MP/M 1.Ø. If more than one console is active each console must consent to resetting of the disk system. This is done by displaying the following message on each console, and then a reset is performed only if all terminals respond with affirmative.

```
 Confirm disk system reset (Y/N) ?
```

6.3.15 Select Disk

6.3.16 Open File

6.3.17 Close File

6.3.18 Search First

6.3.19 Search Next

6.3.20 Delete File

6.3.21 Read Record

6.3.22 Write Record

6.3.23 Make File

6.3.24 Rename File

6.3.25 Interrogate Login

6.3.26 Interrogate Disk

6.3.27 Set DMA Address

6.3.28 Interrogate Allocation

6.3.29 Write Protect Assiigned Disk

6.3.30 Interrogate R/O Bit Vector

6.3.31 Set File Attributes

6.3.32 Get Address of Disk Params

6.3.33 Set/Get User Code

6.3.34 Read Random

6.3.35 Write Random

6.3.36 Compute File Size

6.3.37 Set Random Record

6.4 Description of Extended Disk Operating System

Access to the extended disk operating system (XDOS) facilities is accomplished by passing a function number and information address to the XDOS. In general, the function number is passed in Register C, while the information address is passed in Register pair D,E. Note that this conforms to the PL/M Conventions for parameter passing, and thus the following PL/M procedure is sufficient to link to the XDOS when a value is returned:

```
MON2:        /* XDOS FUNCTION */
  PROCEDURE (FUNC, INFO) BYTE EXTERNAL;
    DECLARE FUNC BYTE;
    DECLARE INFO ADDRESS;
  END MON2;
```

or

```
MON1:   /* XDOS PROCEDURE */
  PROCEDURE (FUNC, INFO) EXTERNAL;
    DECLARE FUNC BYTE;
    DECLARE INFO ADDRESS;
  END MON1;
```

if no returned value is expected.

6.4.1 Absolute Memory Request

The purpose of the absolute memory request operation is to allocate an absolute block of memory specified by the passed memory descriptor parameter. This function allows non-relocatable programs, such as CP/M 2.Ø *.COM files based at the absolute TPA address of Ø1ØØH, to run in the MP/M 1.Ø environment. The single passed parameter is the address of a memory descriptor. The memory descriptor contains four bytes: the memory segment base page address, the memory segment page size, the memory segment attributes, and bank. The only parameters required are the base and size, the other parameters are filled in by XDOS. The operation returns a "boolean" indicating whether or not the allocation was made. A returned value of FFH indicates failure to allocate the requested memory and a value of Ø indicates success. Note that base and size specify base page address and page size where a page is 256 bytes.

The following example illustrates a request for 32K of memory based at location ØØØØH:

```
DECLARE MEMORY$DESCIPTOR STRUCTURE (
  BASE BYTE,
  SIZE BYTE,
  ATTRIBS BYTE,
  BANK BYTE )  INITIAL (ØØH,8ØH,Ø,Ø);

IF MON2 (128,.MEMORY$DESCRIPTOR) THEN
```

```
DO;
   PRINTERROR ('Absolute memory request failed.');
  ...
END;
```

6.4.2 Relocatable Memory Request

The purpose of the relocatable memory request operation is to allocate the requested contiguous memory pages to the calling program. The single passed parameter is the address of the memory descriptor. The only memory descriptor parameter entered by the calling program is the size, the other parameters, base. attributes and size, are filled in by the memory allocation procedure. The operation returns a boolean indicating whether or not the memory request could be satisifed. A returned value of FFH indicates a failure to satisfy the request. Note that base and size specify base page address and page size where a page is 256 bytes.

```
DECLARE MEMORY$DESCIPTOR STRUCTURE (
  BASE BYTE,
  SIZE BYTE,
  ATTRIBS BYTE,
  BANK BYTE )  INITIAL (0,40H,0,0);

IF MON2 (129,.MEMORY$DESCRIPTOR) THEN
DO;
   PRINTERROR ('Relocatable memory request failed.');
  ...
END;
```

6.4.3 Memory Free

The purpose of the memory free operation is to release the specified memory segment back to the operating system. The passed parameter is the address of a memory descriptor. Nothing is returned as a result of this operation.

```
CALL MON1 (130,BUFFERADR);
```

6.4.4 Poll

The purpose of the poll operation is to poll the specified device until a ready condition is received. The calling process relinquishes the processor until the poll is satisfied, allowing other processes to execute.

The following code could be used to implement a generalized console input in a polled system:

```
CONIN:
  PROCEDURE (CONSOLE) BYTE REENTRANT;
```

```
      DECLARE CONSOLE BYTE;

      CALL MON1 (13Ø,CONSOLE);
      DO CASE CONSOLE;
        /* CONSOLE 1 */
        DO;
          ...
          RETURN INPUT(CNS1);
        END;

        ...

        /* CONSOLE n */
        DO;
          ....
          RETURN INPUT(CNSn);
        END;
      END; /* CASE */
    END CONIN;
```

6.4.5 Flag Wait

The purpose of the flag wait operation is to cause a process to relinquish the processor until the flag specified in the call is set. The flag wait operation is used in an interrupt driven system to cause the calling process to ′wait′ until a specific interrupt condition occurs.

The generalized console input example used in poll (section 6.4.4) could be re-written as follows:

```
  CONIN:
    PROCEDURE (CONSOLE) BYTE REENTRANT;
      DECLARE CONSOLE BYTE;


      CALL MON1 (132,CONSOLE);
      DO CASE CONSOLE;
        /* CONSOLE 1 */
        DO;
          RETURN INPUT(CNS1);
        END;

        ...

        /* CONSOLE n */
        DO;
          RETURN INPUT(CNSn);
        END;
      END; /* CASE */
    END CONIN;
```

6.4.6 Flag Set

The purpose of a flag set operation is to wakeup a waiting process. The flag set operation is usually performed by an interrupt service routine after servicing an interrupt and determining which flag is to be set.

The following example shows an interrupt service routine for a system with vectored interrupts and a single device interrupting on a specific level. The reader is referred to section 6.2.8 for an assembly language example of the flag set operation in a flat interrupt system.

```
DISKINT:
  PROCEDURE INTERRUPT 3;

    CALL MON1 (133,DISKINTRPT);
  END DISKINT;
```

6.4.7 Make Queue

The purpose of the make queue operation is to setup a queue control block. A queue is configured as either circular or linked depending upon the message size. Message sizes of 0 to 2 bytes use circular queues while message sizes of 3 or more bytes use linked queues.

A single parameter is passed to make a queue, the queue control block address. The queue control block must contain the queue name, message length, number of messages, and sufficient space to accomodate the messages (and links if the queue is linked).

The following example illustrates how to setup a queue control block for a circular queue with 80 messages of a one byte length.

```
DECLARE CIRCULAR$QUEUE STRUCTURE (
  QL ADDRESS,
  NAME(8) BYTE,
  MSGLEN ADDRESS,
  NMBMSGS ADDRESS,
  DQPH ADDRESS,
  NQPH ADDRESS,
  MSG$IN ADDRESS,
  MSG$OUT ADDRESS,
  MSG$CNT ADDRESS,
  BUFFER (80) BYTE         )
  INITIAL (0,'CIRCQUE ',1.80);

...
...

RET = MON2 (134,.CIRCULAR$QUEUE);
```

The elements of the circular queue shown above are defined as follows:

```
QL      = 2 byte link, set by system
NAME    = 8 ASCII character queue name,
          set by user
MSGLEN  = 2 bytes, length of message,
          set by user
NMBMSGS = 2 bytes, number of messages,
          set by user
DQPH    = 2 bytes, DQ process head,
          set by system
NQPH    = 2 bytes, NQ process head,
          set by system
MSG$IN  = 2 bytes, pointer to next
          message in, set by system
MSG$OUT = 2 bytes, pointer to next
          message out, set by system
MSG$CNT = 2 bytes, number of messages
          to be read, set by system
BUFFER  = n bytes, where n is equal to
          the message length times the
          number of messages, space
          allocated by user, set by system

Queue Overhead     = 24 bytes
```

The following example illustrates how to setup a queue control block for a linked queue containing 4 messages, each 33 bytes in length:

```
DECLARE LINKED$QUEUE STRUCTURE (
  QL ADDRESS,
  NAME (8) BYTE,
  MSGLEN ADDRESS,
  NMBMSGS ADDRESS,
  DQPH ADDRESS,
  NQPH ADDRESS,
  MH ADDRESS,
  MT ADDRESS,
  BH ADDRESS,
  BUFFER (140) BYTE   )
  INITIAL (0,'LNKQUE  ',33,4);

...
...

RET = MON2 (134,.LINKED$QUEUE);
```

The elements of the linked queue shown above are defined as follows:

```
QL      = 2 byte link, set by system
NAME    = 8 ASCII character queue name,
          set by user
```

MSGLEN	= 2	bytes, length of message, set by user
NMBMSGS	= 2	bytes, number of messages, set by user
DQPH	= 2	bytes, DQ process head, set by system
NQPH	= 2	bytes, NQ process head. set by system
MH	= 2	bytes, message head, set by system
MT	= 2	bytes, message tail, set by system
BH	= 2	bytes, buffer head, set by system
BUFFER	= n	bytes where n is equal to the message length plus two, times the number of messages, space allocated by the user, set by the system

6.4.8 Open Queue

The purpose of the open queue operation is to place the actual queue control block address into the user queue control block. The result of this operation is that a user program can obtain access to system queues by knowing only the queue name. the actual address of the queue itself is obtained as a result of opening the queue.

Once a queue has been opened, the queue may be read from or written to using the queue read and write operations. The MSGADR field of the user queue control block is the address of a local user buffer. When a read queue operation is performed data is placed at the buffer pointed to by MSGADR. When a write queue operation is performed data is written into the actual queue from the data in the buffer pointed to by MSGADR.

The operation returns a boolean indicating whether or not the open queue operation found the queue to be opened. A returned value of ØFFH indicates failure while a zero indicates success.

The following example illustrates the opening of the "SPOOL" queue:

```
DECLARE USER$QUEUE$CONTROL$BLOCK STRUCTURE (
  POINTER ADDRESS,
  MSGADR ADDRESS,
  NAME (8) BYTE   )
   INITIAL (Ø,.BUFFER,'SPOOL   ');

DECLARE BUFFER (33) BYTE;

...
...
```

```
 RET = MON2 (135,.USER$QUEUE$CONTROL$BLOCK);
```

The elements of the user queue control block shown above are defined as follows:

```
 POINTER = 2 bytes, set by system to address of
             actual queue
 MSGADR  = 2 bytes, address of user buffer,
             set by user
 NAME    = 8 bytes, ASCII queue name,
             set by user
```

6.4.9 Delete Queue

The purpose of the delete queue operation is to remove the specified queue from the queue list. A single parameter is passed to delete a queue, the address of the actual queue. This value can be obtained from the POINTER field of a currently open user queue control block.

The operation returns a boolean indicating whether or not the delete queue operation found the queue and deleted it. A returned value of ØFFH indicates failure while a zero indicates success.

The following example illustrates the deletion of the "TEMPQUE " queue:

```
 DECLARE USER$QCB STRUCTURE (
   POINTER ADDRESS,
   MSGADR ADDRESS,
   NAME (8) BYTE     )
   INITIAL (Ø,.BUFFER,'TEMPQUE ');

 DECLARE BUFFER (16) BYTE;

 ...
 ...

 RET = MON2 (136,USER$QCB.POINTER);
```

6.4.1Ø Read Queue

The purpose of the read queue operation is to read a message from a specified queue. If no message is available at the queue the calling process relinquishes the processor until a message is posted at the queue. The single passed parameter is the address of a user queue control block. When a message is available at the queue, it is copied into the buffer pointed to by the MSGADR field of the user queue control block.

The following example illustrates the read queue operation:

```
 DECLARE USER$QCB STRUCTURE (
   POINTER ADDRESS,
   MSGADR ADDRESS,
   NAME (8) BYTE   )
   INITIAL (Ø,.BUFFER,'DATAQUE ');

 DECLARE BUFFER (8Ø) BYTE;

 ...
 ...

 CALL MON1 (137,.USER$QCB);
```

6.4.11 Conditional Read Queue

The purpose of the conditional read queue operation is to read a message from a specified queue if a message is available. The single passed parameter is the address of a user queue control block. If a message is available at the queue, it is copied into the buffer pointed to by the MSGADR field of the user queue control block.

The operation returns a boolean indicating whether or not a message was available at the queue. A returned value of ØFFH indicates no message while a zero indicates that a message was available and that it was copied into the user buffer.

The following example illustrates the conditional read queue operation:

```
 DECLARE USER$QCB STRUCTURE (
   POINTER ADDRESS,
   MSGADR ADDRESS,
   NAME (8) BYTE   )
   INITIAL (Ø,.BUFFER,'DATAQUE ');

 DECLARE BUFFER (8Ø) BYTE;

 ...
 ...

 RET = MON2 (138,.USER$QCB);
```

6.4.12 Write Queue

The purpose of the write queue operation is to write a message to a specified queue. If no buffers are available at the queue the calling process relinquishes the processor until a buffer is available at the queue. The single passed parameter is the address of a user queue control block. When a buffer is available at the queue, the buffer pointed to by the MSGADR field of the user queue control block is copied into the actual queue.

The following exam   the write queue operation:

```
 DECLARE USER$QCB STR
   POINTER ADDRESS,
   MSGADR ADDRESS,
   NAME (8) BYTE   )
   INITIAL (0,.BUFFER,

 DECLARE BUFFER (80) BY

 ...
 ...

 CALL MON1 (139,.USER$QCB);
```

6.4.13 Conditional Write Qu

The purpose of the condi   ite queue operation is to
write a message to a spec   e if a buffer is available.
The single passed parameter   address of a user queue
control block. If a b   available at the queue, the
buffer pointed to by the MS   of the user queue control
block is copied into the ac   e.

The operation returns a   dicating whether or not a
buffer was available at   e. A returned value of 0FFH
indicates no buffer while a   cates that a buffer was
available and that the user   s copied into it.

The following example il   the conditional write queue
operation:

```
 DECLARE USER$QCB STRUCTURE
   POINTER ADDRESS,
   MSGADR ADDRESS,
   NAME (8) BYTE   )
   INITIAL (0,.BUFFER,'DATA

 DECLARE BUFFER (80) BYTE;

 ...
 ...

 RET = MON2 (140,.USER$QCB);
```

6.4.14 Delay

The purpose of the delay   is to delay execution of
the calling process for   ified number of system time
units. Use of the delay op   ids the typical programmed
delay loop. It allows othe   es to use the processor
while the specified peri   e elapses. The system time
unit is typically 60 Hz   liseconds) but may vary
according to application.   ample it is likely that in

Europe it would be 50 Hz (20 milliseconds).

The delay is specifed as a 16-bit integer. Since calling the delay procedure is usually asynchronous to the actual time base itself, there is some degree of uncertainty in the exact amount of time delayed. Thus a delay of 10 ticks gaurantees a delay of at least 10 ticks, but it may be as big as almost 11 ticks.

In the following example the delay operation is used in a situation where a peripheral device, such as a CRT, requires a delay of NULL$TIME following a line feed.

```
IF CHAR = LF THEN
DO;
  CALL MON1 (141,NULL$TIME);
 END;
```

6.4.15 Dispatch

The purpose of the dispatch operation is to allow the operating system to determine the highest priority ready process and then to give it the processor. This call is provided in XDOS to allow systems without interrupts the capability of sharing the processor amoung compute bound processes. Since all user processes usually run at the same priority, invoking the dispatch operation at various points in a program will allow other users to obtain the processor in a round-robin fashion.

Dispatch is intended for non-interrupt driven environments in which it is desirable to enable a compute bound process to relinquish the use of the processor.

```
...
CALL MON1 (142,0);
..
```

Note the dummy parameter of 0 in the MON1 call.

6.4.16 Terminate Process

The purpose of the terminate process operation is to terminate the calling process. A single passed parameter indicates whether or not the process should be terminated if it is a system process. A 0FFH indicates that the process should be unconditionally terminated, a zero indicates that only a user process is to be deleted. There are no results returned from this operation, the calling process simply ceases to exist as far as MP/M is concerned.

The following example illustrates the terminate process operation:

```
EXIT:
  PROCEDURE;
```

```
      CALL MON1 (143,0);
    END EXIT;

  ...

  CALL EXIT;
```

6.4.17 Create Process

The purpose of the create process operation is to create one or more processes by placing the passed process descriptors on the MP/M ready list.

A single parameter is passed, the address of a process descriptor. The first field of the process descriptor is a link field which may point to other process descriptors.

The following example illustrates the creation of two processes which execute the same piece of reentrant code:

```
  DECLARE CNS$HNDLR$1 STRUCTURE (
    PL ADDRESS,
    STATUS BYTE,
    PRIORITY BYTE,
    STKPTR ADDRESS,
    NAME (8) BYTE,
    CONSOLE BYTE,
    MEMSEG BYTE,
    B ADDRESS,
    THREAD ADDRESS,
    DISK$SET$DMA ADDRESS,
    DISK$SLCT BYTE,
    DCNT ADDRESS,
    SEARCHL BYTE,
    SEARCHA ADDRESS,
    SCRATCH (2) BYTE )
    INITIAL (.CNS$HNDLR$2,0,200,.CNS$1$STK(23),
             'CNS1    ',1);

  DECLARE CNS$HNDLR$2 STRUCTURE (
    PL ADDRESS,
    STATUS BYTE,
    PRIORITY BYTE,
    STKPTR ADDRESS,
    NAME (8) BYTE,
    CONSOLE BYTE,
    MEMSEG BYTE,
    B ADDRESS,
      THREAD ADDRESS,
    DISK$SET$DMA ADDRESS,
    DISK$SLCT BYTE,
    DCNT ADDRESS,
    SEARCHL BYTE,
          SEARCHA ADDRESS,
          SCRATCH (2) BYTE )
          INITIAL (Ø,Ø,2ØØ,.CNS$2$STK(23),
                   'CNS2    ',2);

        DECLARE CNS$1$STK (23) ADDRESS;
        DECLARE CNS$2$STK (23) ADDRESS;

        ...
        ...

        CNS$1$STK(23) = .CNS$HNDLR;
        CNS$2$STK(23) = .CNS$HNDLR;
        CALL MON1 (144,.CNS$HNDLR$1);
        ...

        CNS$HNDLR:
          PROCEDURE REENTRANT;
            DECLARE BUFFER (8Ø) BYTE;

            ...
            ...

            DO FOREVER;

              ...

            END;
          END CNS$HNDLR;
```

The elements of the process descriptor shown above are defined as follows:

PL	= 2 byte link field, initially set by user to address of next process descriptor, or zero if no more
STATUS	= 1 byte, process status, set by system
PRIORITY	= 1 byte, process priority, set by user
STKPTR	= 2 bytes, stack pointer, initially set by user
NAME	= 8 bytes, ASCII process name, set by user
CONSOLE	= 1 byte, console to be used by process, set by user
MEMSEG	= 1 byte, memory segment table index
B	= 2 bytes, system scatch area
THREAD	= 2 bytes, process list thread, set by system
DISK$SET$DMA	= 2 bytes, default DMA address, set by user
DISK$SLCT	= 1 byte, default disk
DCNT	= 2 bytes, system scratch byte
SEARCHL	= 1 byte, system scratch byte
SEARCHA	= 2 bytes, system scratch bytes
SCRATCH	= 2 bytes, system scratch bytes

6.4.18 Set Priority

The purpose of the set priority operation is to set the priority of the calling process to that of the passed parameter. This function is useful in situations where a process needs to have a high priority during an initialization phase. but after that is to run at a lower priority.

A single passed parameter contains the new process priority. There are no results returned from setting priority.

The following example illustrates setting the priority to 2ØØ:

```
...
CALL MON1 (145,2ØØ);
...
```

6.4.19 Attach Console

The purpose of the attach console operation is to attach the console specified in the CONSOLE field of the process descriptor to the calling process. If the console is already attached, the calling process relinquishes the processor until the console is detached and the calling process is the highest priority process waiting for the console.

There are no passed parameters and there are no returned results.

The following example illustrates the code to attach the console:

```
...
CALL MON1 (146,Ø);
...
```

6.4.2Ø Detach Console

The purpose of the detach console operation is to detach the console specified in the CONSOLE field of the process descriptor from the calling process. If the console is not currently attached, no action takes place.

There are no passed parameters and there are no returned results.

The following example illustrates the code to detach the console:

```
...
CALL MON1 (147,Ø);
```

```
...
```

6.4.21 Set Console

The purpose of the set console operation is to detach the currently attached console and then to attach the console specified as a calling parameter. If the console to be attached is already attached to another process descriptor, the calling process relinquishes the processor until the console is available.

A single passed parameter contains the console number to be attached. There are no returned results.

The following example illustrates the code to attach console 4:

```
...
CALL MON1 (148,4);
...
```

6.4.22 Assign Console

The purpose of the assign console operation is to directly assign the console to a specified process. This assignment is done regardless of whether or not the console is currently attached to some other process. A single parameter is passed to assign console which is the address of a data structure containing the console number for the assignment, an 8 character ASCII process name, and a boolean indicating whether or not a match with the console field of the process descriptor is required (true or ØFFH indicates it is required).

The operation returns a boolean indicating whether or not the assignment was made. A returned value of ØFFH indicates failure to assign the console, either because a process descriptor with the specified name could not be found, or that a match was required and the console field of the process descriptor did not match the specified console. A returned value of zero indicates a successful assignment.

The following example illustrates the assignment of console 3 to a process named ´DISPLAY´.

```
DECLARE ASSIGN$PARAMS STRUCTURE (
  CONSOLE BYTE,
  NAME (8) BYTE,
  MATCH$REQD BYTE )
  INTIAL (
  3,´DISPLAY ´,ØFFH);

...
...
```

```
 RET = MON2 (149,.ASSIGN$PARAMS);
```

6.4.23 Send CLI Command

The purpose of the send CLI command operation is to permit running programs to send command lines to the Command Line Interpreter. A single parameter is passed which is the address of a data structure containing the default disk/user code, console and command line itself. There are no results returned to the calling process.

The following example illustrates a command sent to spool a list file. Note that the command line must be terminated with a null.

```
 DECLARE CLI$COMMAND STRUCTURE (
   DEFAULT$DISK BYTE,
   CONSOLE BYTE,
   COMMAND (80) BYTE )
    INITIAL (
   0,1,'SPOOL ANALYSIS.LST',0);

 ...
 ...

 CALL MON1 (150,.CLI$COMMAND);
```

6.4.24 Call Resident System Procedure

The purpose of the call resident system procedure operation is to permit programs to call the optional resident system procedures. A single passed parameter contains the address of a call parameter block data structure which contains the address of a resident system procedure name and a parameter to be passed to the resident system procedure.

The operation returns a 1 if the resident system procedure called is not present, otherwise it returns the code passed back from the resident system procedure. Typically a returned value of FFH indicates failure while a zero indicates success.

```
 DECLARE CPB STRUCTURE (
   RSP$NAME$ADR ADDRESS,
   RSP$PARAM ADDRESS )
   INITIAL (
   .RSP$NAME,0);

 DECLARE RSP$NAME (8) BYTE
   INITIAL ('CONVERT ');

 ...
 ...
```

```
CPB.RSP$PARAM = VALUE;
RET = MON2 (151,.CPB);
```

6.4.25 Parse Filename

The purpose of the parse filename operation is to prepare a file control block from an input ASCII string containing a file name. A single parameter is the address of a data structure which contains the address of the ASCII file name string and the address of the target file control block.

The operation returns an FFFFH if the input ASCII string contains an invalid file name. A zero is returned if the ASCII string contains a single file name, otherwise the address of the first character following the file name is returned.

```
DECLARE PFCB STRUCTURE (
  FILE$NAME$ADR ADDRESS,
  FCB$ADR ADDRESS )
  INITIAL (
  .FILE$NAME,.FCB);

DECLARE FILE$NAME (80) BYTE;

DECLARE FCB (33) BYTE;

...
...

RET = MON2 (152,.PFCB);
```

6.4.26 Get Console Number

The purpose of the get console number operation is to obtain the value of the console field from the process descriptor of the calling program. There are no passed parameters and the returned result is the console number of the calling process.

```
CONSOLE = MON2 (153,0);
```

6.4.27 System Data Address

The purpose of the system data address operation is to obtain the base address of the system data page. The system data page resides in the top 256 bytes of available memory. It contains configuration information used by the MP/M loader as well as run time data including the submit flags. There are no passed parameters and the returned result is the base address of the system data page.

```
DECLARE SYS$DAT$PG$ADR ADDRESS;
DECLARE SYS$DAT$PG BASED SYS$DAT$PG$ADR (256) BYTE;

...
...

SYS$DAT$PG$ADR = MON2 (154,0);
```

6.4.28 Get Date and Time

The purpose of the get date and time operation is to obtain the current encoded date and time. A single passed parameter is the address of a data structure which is to contain the date and time. The date is represented as a 16-bit integer with day 1 corresponding to January 1, 1978. The time is respresented as three bytes: hours, minutes and seconds, stored as two BCD digits.

```
DECLARE TOD STRUCTURE (
  DATE ADDRESS,
  HRS BYTE,
  MIN BYTE,
  SEC BYTE );

...
...

CALL MON1 (155,.TOD);
```

6.5 Console Commands / Operator Interface

The purpose of this section is to describe the console commands which make up the operator interface to the MP/M 1.0 operating system. It is important to note from the outset that there are no system defined or built-in commands. That is, the system has no reserved or special commands. All commands in the system are a reflection of resident system processes specified during system generation or programs residing on disk in either the CP/M 2.0 *.COM file format or in the MP/M *.PRL (page relocatable) format.

6.5.1 Run Program

A program is run by typing in the program name followed by a carriage return, <cr>. Some programs obtain parameters on the same line following the program name. Characters on the line following the program name constitute what is called the command tail. The command tail is copied into location 0080H (relative to the base of the memory segment in which the program resides) by the Command Line Interpreter (CLI). The CLI also parses the command tail producing two file control blocks at 005CH and 006CH respectively.

The programs which are provided with MP/M 1.0 are described in sections 6.6 and 6.7.

6.5.2 Abort Program

A program may be aborted by typing a control C (^C) at the console. In order for a program to be aborted it is necessary that it check console status to obtain the ^C. An alternate solution is to implement a BIOS with a "live console". That is. to write a process which continually monitors the console input, passing normal input data on to the conin procedure and taking appropriate action when a ^C is detected.

6.5.3 Run Resident System Process

At the operator interface there is no difference between running a program from disk and running a resident system process. The actual difference is that resident system processes do not need to be loaded from disk because they are loaded by the MP/M loader when a system cold start is performed and remain resident.

A brief description of the CLI operation should illustrate this point. When the CLI receives a command line it parses the first entry on the command line and then tries to open a queue using the parsed name. If the open queue succeeds the command tail is written to the queue and the CLI operation is finished. If the open queue fails a file type of PRL is entered for the parsed file name and a file open is attempted. If the file open succeeds then the header of the PRL file is read to determine the memory requirements. A relocatable memory request is made to obtain a memory segment in which to load and run the program.

If this request is satisfied the PRL file is read into the memory segment, relocated, and it is executed, completing the CLI operation.

If the PRL file type open fails then the file type of COM is entered for the parsed file name and a file open is attempted. If the open succeeds then a memory request is made for the absolute TPA, memory segment based at 0000H. If this request is satisfied the COM file is read into the absolute TPA and it is executed, completing the CLI operation.

6.5.4 Detach Program

There are two methods for detaching from a running program. The first is to type a control D (^D) at the console. The second method is for a program to make an XDOS detach call.

The restriction on the former method, typing ^D, is that the running program must be performing a check console status to observe the detach request. This requirement is removed if a "live console" BIOS has been implemented as described in section 6.5.2.

6.5.5 Attach Program

A program which is waiting for the console, such as a detached program, may be attached to the console by typing 'ATTACH' followed by the program name. A program may only be attached from the console at which it was detached. If the TMP has ownership of the console and the user enters a ^D, the next highest priority ready process which is waiting for the console begins running.

6.5.6 Line Editing and Output Control

The Terminal Message Process (TMP) allows certain line editing functions while typing in command lines:

rubout	Delete the last character typed at the console, removes and echoes the last character
ctl-C	MP/M abort program. Terminate running process.
ctl-E	Physical end of line.
ctl-H	Delete the last character typed at the console, backspaces one character position.
ctl-J	(line feed) terminate current input.
ctl-M	(carriage return) terminates input.
ctl-R	Retype current command line: types a "clean line" following character deletion with rubouts.

ctl-U	Remove current line after new line.
ctl-X	Delete the entire line typed at the console, backspaces to the beginning of the current line
ctl-Z	End input from the console.

The control functions ctl-P and ctl-S affect console output as shown below.

ctl-P	Copy all subsequent console output to the list device. Output is sent to both the list device and the console device until the next ctl-P is typed.
ctl-S	Stop the console output temporarily. Program execution and output continue when the next character is typed at the console (e.g., another ctl-S). This feature is used to stop output on high speed consoles, such as CRT's, in order to view a segment of output before continuing.

6.6 Commonly Used System Programs

The commonly used system programs (CUSPs) or transient commands, as they are called in CP/M 2.0, are loaded from the currently logged disk and executed in a relocatable memory segment if their type is PRL or in the absolute TPA if their type is COM.

This section contains a brief description of the CUSPs. Operation of many of the CUSPs is identical to CP/M 2.0. In these cases, the reader is referred to the Digital Research document titled "An Introduction to CP/M Features and Facilities" for a complete description of the CUSP.

6.6.1 Get/Set User Code

The USER command is used to display the current user code as well as to set the user code value.

6.6.2 Erase File

The ERA (erase) command removes specified files from the currently logged-in disk.

6.6.3 Type File

The TYPE command displays the contents of the specified ASCII source file on the console device. The TYPE command expands tabs (ctl-I characters), assuming tab positions are set at every eighth column.

6.6.4 File Directory

The DIR (directory) command causes the names of files on the logged-in disk to be listed on the console device. If no files can be found on the selected diskette which satisfy the directory request, then the message "Not found" is typed at the console.

6.6.5 Rename File

The REN (rename) command allows the user to change the name of files on disk.

6.6.6 Text Editor

The ED (editor) command allows the user to edit ASCII text files.

6.6.7 Peripheral Interchange Program

The PIP (peripheral interchange program) command allows the user to perform disk file and peripheral transfer operations.

6.6.8 Assembler

The ASM (assembler) command allows the user to assemble the specified program on disk.

6.6.9 Submit

The SUBMIT command allows the user to submit a file of commands for batch processing.

6.6.10 Status

The STAT (status) command provides general statistical information about the file storage.

6.6.11 Dump

The DUMP command types the contents of the specified disk file on the console in hexadecimal form.

6.6.12 Hexcom (Load)

The HEXCOM command reads the specified disk file of type HEX and produces a memory image file of type COM which can subsequently be executed.

6.6.13 Concat

The CONCAT command concatenates source files to produce a single destination file. The form of the command tail is as follows:

```
 CONCAT dest.typ=src1.typ,src2.typ,src3.typ,...
```

6.6.14 Genmod

The GENMOD command accepts a file which contains two concatenated files of type HEX which are offset from each other by 0100H bytes, and produces a file of type PRL (page relocatable).

6.6.15 Dynamic Debugging Tool

The DDT (dynamic debugging tool) command loads and executes the MP/M debugger.

6.7 Extended System Programs

The extended system programs (ESPs) are new programs specifically designed to facilitate use of the MP/M operating system. The ESPs may either be resident on disk as files of the PRL type, or they may be resident system processes. Resident system processes are selected at the time of system generation.

6.7.1 System Status

The MPMSTAT command allows the user to display the run-time status of the MP/M operating system. MPMSTAT is invoked by typing 'MPMSTAT' followed by a <cr>. A sample MPMSTAT output is shown below:

```
****** MP/M 1.Ø Status Display ******
Ready Process(es):
  MPMSTAT  cli      Idle
Process(es) DQing:
 [SCHED   ] Sched
 [ATTACH  ] ATTACH
 [SPOOL   ] Spool
Process(es) NQing:
Delayed Process(es):
Polling Process(es):
  PIP
Swapped Process(es):
Process(es) Flag Waiting:
  Ø1 -  Tick
  Ø2 -  Clock
Flag(s) Set:
  Ø3
Queue(s):
  tod      SCHED    ATTACH   STOPSPLR SPOOL    SYSTAT   Cliq
  ParseQ   ListMQ   DiskMQ
Process(es) Attached to Consoles:
  [Ø] - MPMSTAT
  [1] - PIP
Process(es) Waiting for Consoles:
  [Ø] - TMPØ     DIR
  [1] - TMP1
Memory Allocation:
  Base = ØØØØH  Size = 4ØØØH  Allocated to PIP      [1]
  Base = 4ØØØH  Size = 2ØØØH  * Free *
  Base = 6ØØØH  Size = 11ØØH  Allocated to DIR      [Ø]
```

6.7.2 Spooler

The SPOOL command allows the user to spool ASCII text files to the list device. Multiple file names may be specified in the command tail. The spooler expands tabs (ctl-I characters), assuming tab positions are set at every eighth column.

The spooler queue can be purged at any time by using the

STOPSPLR command.

An example of the SPOOL command is shown below:

```
SPOOL LOAD.LST LETTER.PRN
```

6.7.3 Time and Date

The TOD (time of day) command allows the user to read and set the date and time. Entering 'TOD' followed by a <cr> will cause the current date and time to be displayed on the console. Entering 'TOD' followed by a date and time will set the date and time when a <cr> is entered following the prompt to strike a key. Each of these TOD commands are illustrated below:

```
TOD <cr>
Wed 09/15/79 09:15:37

-or-

TOD 9/20/79 10:30:00
 Strike key to set time
 Thu 09/20/79 10:30:00
```

6.7.4 Scheduler

The SCHED (scheduler) command allows the user to schedule a program for execution. Entering 'SCHED' followed by a date, time and command line will cause the command line to be executed when the specified date and time is reached.

In the example shown below, the program 'SAMPLE' will be loaded from disk and executed on September 18, 1979 at 10:30 PM.

```
SCHED 9/18/79 22:30 SAMPLE
```

7.Ø System Generation and Loading

MP/M 1.Ø system generation consists of the preparation of a system data file and the concatenation of both required and optional code files to produce a file named 'MPM.SYS'. The operation is performed using a GENSYS program which can be run under either MP/M 1.Ø or CP/M 2.Ø. The GENSYS automates the system generation process by prompting the user for optional arameters and then prepares the 'MPM.SYS' file.

MP/M 1.Ø system loading consists of reading in the 'MPM.SYS' file and relocating the entire operating system into the position designated by the system data portion of 'MPM.SYS'. The MP/M 1.Ø loader can be run under CP/M 2.Ø making it possible to debug MP/M 1.Ø system programs while running under a CP/M 2.Ø debugger.

7.1 MP/M System File Components

The MP/M system file, 'MPM.SYS' consists of four components: the system data page, the customized BIOS, the MP/M nucleus, and the resident system processes.

7.1.1 System Data

The system data page contains 256 bytes used by the loader to dynamically configure the system. The system data page can be prepared using the GENSYS program or it can be manually prepared using DDT or SID. The following table describes the byte assignments:

Byte	Assignment
ØØØ-ØØØ	Top page of memory
ØØ1-ØØ1	Number of consoles
ØØ2-ØØ2	Breakpoint restart number
ØØ3-ØØ3	Allocate stacks for user system calls
ØØ4-Ø15	Unassigned
Ø16-Ø31	Memory segment table, a list of base page addresses in ascending order terminated by a ØFFH.
Ø32-Ø47	Memory segment bank corresponding to memory segment table entry.
Ø48-Ø79	Breakpoint vector table, filled in by DDTs
Ø8Ø-111	Stack addresses for user system calls
112-127	Unassigned
128-143	Submit flags

7.1.2 Customized BIOS

The customized BIOS is obtained from a file named 'BIOS.SPR'. The 'BIOS.SPR' file is actually a file of type PRL containing the page relocatable version of the user customized BIOS. A submit file on the distribution diskette named 'MACPRL.SUB' can be used to generate the user customized BIOS. The following sequence of commands will produce a 'BIOS.SPR' file given a user 'BIOS.ASM' file:

```
 SUBMIT MACPRL BIOS
 REN BIOS.SPR=BIOS.PRL
```

7.1.3 Nucleus

The MP/M nucleus file named 'MPM.SPR' is a page relocatable file containing the priority driven MP/M nucleus. The nucleus contains the following code pieces: root module. XDOS interface, dispatcher, queue management, flag management, memory management, terminal handler, terminal message process, command line interpreter, file name parser, time base management, BDOS and BDOS interface.

7.1.4 Resident System Processes

Resident system processes are identified by a file type of RSP. The RSP files distributed with MP/M 1.0 include: MP/M run-time system status display, printer spooler, time and date conversion, and a scheduler.

At system generation time the user is prompted to select which RSPs are to be concatenated to the 'MPM.SYS' file.

It is possible for the user to prepare custom resident system processes. The resident system processes must follow these rules:

* The file itself must be page relocatable. Page relocatable files can be simply generated using the submit file 'MACPRL.SUB'.

* The first two bytes of the resident system process are reserved for the address of the BDOS. Thus a resident system process can access the BDOS by loading the two bytes at relative 0000-0001H and then performing a PCHL.

* The process descriptor for the resident system process must begin at the third byte position. The contents of the process descriptor are described in section 6.4.17.

7.2 Gensys

The GENSYS program is used to prepare the 'MPM.SYS' file for MP/M from a system data file and concatenation of both required and optional code files. GENSYS can be run under either CP/M 2.0 or MP/M 1.0.

The operation of GENSYS is best illustrated with the sample execution shown below:

```
 GENSYS
```

```
MP/M 1.0 System Generation
==========================

Top page of memory = D0
Number of consoles = 2
Breakpoint RST #   = 5
Allocate user stacks for system calls (Y/N)? y
Memory segment bases, (ff terminates list)
 : 00
 : 40
 : 60
 : ff
Select Resident System Processes: (Y/N)
TIME     ? y
SCHED    ? n
ATTACH   ? y
Spool    ? y
MPMSTAT  ? y
```

7.3 Loader

The MPMLDR program loads the 'MPM.SYS' file and dynamically relocates and configures the MP/M 1.0 operating system. MPMLDR can be run under either CP/M 2.0, providing that the top page of memory is set below the resident CP/M 2.0 and debugger, or loaded from the first two tracks of a disk by the bootstrap.

The MPMLDR provides a display of the system loading and configuration. It does not require any operator interaction. In the following example the 'MPM.SYS' file prepared by GENSYS (shown in section 7.2) is loaded:

```
MPMLDR

MP/M 1.0 Loader
===============

Number of consoles = 2
Breakpoint RST #   = 5
Top of memory      = D0FFH

Memory Segment Table:
SYSTEM  DAT  D000H  0100H
CONSOLE DAT  CE00H  0200H
USERSYS STK  CD00H  0100H
BIOS    SPR  C900H  0400H
BDOS    SPR  B800H  1100H
MPM     SPR  9100H  2700H
-------------------------
Memseg  Usr  6000H  3100H
Memseg  Usr  4000H  2000H
Memseg  Usr  0000H  4000H
```

Appendix A: Flag Assignments

Flag Assignments

0	Reserved
1	System time unit tick
2	One second interval
3	One minute interval
	Undefined
	Undefined
15	Undefined

Priority Assignments

0 - 31	Interrupt handlers
32 - 63	System processes
64 - 197	Undefined
198	Teminal message processes
199	Command line interpreter
200 - 254	User processes
255	Idle process

Sample Basic I/O System

This appendix contains a sample BIOS/XIOS. It illustrates the required BIOS support for multiple consoles as well as procedures to support the XIOS calls.

The BIOS must be made into a system page relocatable file (BIOS.SPR) for the generation of the MPM.SYS file by the GENSYS program. The following procedure can be followed to produce a BIOS.SPR file:

* Prepare the customized BIOS with an ORG 0000H. Note that the external jump vector is actually negative (below the origin of 0.

* Assuming a system disk in drive A: and the BIOS.ASM file is on drive B:, enter the commands-

```
A>MAC B:BIOS $PP+S
      ;ASSEMBLE THE BIOS.ASM FILE, LIST WITH SYMBOL TABLE
A>ERA B:BIOS.HX0
A>REN B:BIOS.HX0=B:BIOS.HEX
A>MAC B:BIOS $PZSZ+R
      ;ASSEMBLE THE BIOS.ASM FILE AGAIN OFFSET BY 100H
      ;THE OFFSET IS GENERATED WITH THE +R MAC OPTION
A>PIP B:BIOS.HEX=B:BIOS.HX0[I],B:BIOS.HEX[h]
      ;CONCATENATE THE HEX FILES
A>GENMOD B:BIOS.HEX BIOS.SPR
      ;GENERATE THE RELOCATABLE BIOS.SPR FILE
```

```
                        PAGE    0
                        TITLE   'Basic I/O System'
                        MACLIB  DISKDEF
                ;
                ; BIOS FOR MICRO-2 COMPUTER
                ;
                ;
0000 =          FALSE   EQU     0
FFFF =          TRUE    EQU     NOT FALSE
                ;
0000                    ORG     0000H

                ;
                ;       EXTERNAL JUMP VECTOR, BELOW THE BIOS
FFFD =          PDISP   EQU     $-3             ;MP/M DISPATCHER
FFFA =          XDOS    EQU     PDISP-3         ;BDOS/XDOS ENTRY
                ;
                ;       JUMP VECTOR FOR INDIVIDUAL SUBROUTINES
0000 C34800             JMP     COLDSTART       ;COLD START
                WBOOT:
0003 C34800             JMP     WARMSTART       ;WARM START
0006 C34D00             JMP     CONST           ;CONSOLE STATUS
0009 C35600             JMP     CONIN           ;CONSOLE CHARACTER IN
000C C35F00             JMP     CONOUT          ;CONSOLE CHARACTER OUT
000F C3B400             JMP     LIST            ;LIST CHARACTER OUT
0012 C35D00             JMP     PUNCH           ;PUNCH CHARACTER OUT
0015 C35400             JMP     READER          ;READER CHARACTER OUT
0018 C3BB01             JMP     HOME            ;MOVE HEAD TO HOME
001B C3CC01             JMP     SELDSK          ;SELECT DISK
001E C3F601             JMP     SETTRK          ;SET TRACK NUMBER
0021 C31702             JMP     SETSEC          ;SET SECTOR NUMBER
0024 C32F02             JMP     SETDMA          ;SET DMA ADDRESS
0027 C33502             JMP     READ            ;READ DISK
002A C33A02             JMP     WRITE           ;WRITE DISK
002D C30000             JMP     $-$             ;LIST STATUS
0030 C31D02             JMP     SECTRAN         ;SECTOR TRANSLATE

0033 C33C01             JMP     SELMEMORY       ; SELECT MEMORY
0036 C32101             JMP     POLLDEVICE      ; POLL DEVICE
0039 C33D01             JMP     STARTCLOCK      ; START CLOCK
003C C34301             JMP     STOPCLOCK       ; STOP CLOCK
003F C34801             JMP     EXITREGION      ; EXIT REGION
0042 C34F01             JMP     MAXCONSOLE      ; MAXIMUM CONSOLE NUMBER
0045 C35201             JMP     SYSTEMINIT      ; SYSTEM INITIALIZATION
                ;
                COLDSTART:
                WARMSTART:
0048 0E00               MVI     C,0
004A C3FAFF             JMP     XDOS            ; SYSTEM RESET, TERMINATE PR
                ;
                ;
                ;I/O HANDLERS
                ;
                ;
                ;  MP/M 1.0   CONSOLE BIOS
```

```
                    ;
                    ;
0002 =              NMBCNS  EQU     2           ; NUMBER OF CONSOLES

0083 =              POLL    EQU     131         ; XDOS POLL FUNCTION
008D =              XDELAY  EQU     141         ; XDOS DELAY FUNCTION

0000 =              PLLPT   EQU     0           ; POLL PRINTER
0001 =              PLDSK   EQU     1           ; POLL DISK
0002 =              PLCO0   EQU     2           ; POLL CONSOLE OUT #0
0003 =              PLCO2   EQU     3           ; POLL CONSOLE OUT #1 (PORT 2)
0004 =              PLCI0   EQU     4           ; POLL CONSOLE IN #0
0005 =              PLCI2   EQU     5           ; POLL CONSOLE IN #1 (PORT 2)

                    ;
                    CONST:                      ; CONSOLE STATUS
004D CD6600                 CALL    PTBLJMP     ; COMPUTE AND JUMP TO HNDLR
0050 7C00                   DW      PT0ST       ; CONSOLE #0 STATUS ROUTINE
0052 D900                   DW      PT2ST       ; CONSOLE #1 (PORT 2) STATUS RT

                    READER:                             ; READER NOT IMPLEMENTED
                                                ; *** DEFAULTS TO CONIN #0 ***
0054 1600                   MVI     D,0

                    CONIN:                      ; CONSOLE INPUT
0056 CD6600                 CALL    PTBLJMP     ; COMPUTE AND JUMP TO HNDLR
0059 8400                   DW      PT0IN       ; CONSOLE #0 INPUT
005B E100                   DW      PT2IN       ; CONSOLE #1 (PORT 2) INPUT

                    PUNCH:                      ; PUNCH NOT IMPLEMENTED
                                                ; *** DEFAULTS TO CONOUT #0 ***
005D 1600                   MVI     D,0

                    CONOUT:                             ; CONSOLE OUTPUT
005F CD6600                 CALL    PTBLJMP     ; COMPUTE AND JUMP TO HNDLR
0062 9600                   DW      PT0OUT      ; CONSOLE #0 OUTPUT
0064 F300                   DW      PT2OUT      ; CONSOLE #1 (PORT 2) OUTPUT

                    ;
                    PTBLJMP:                    ; COMPUTE AND JUMP TO HANDLER
                                                ; D = CONSOLE #
                                                ; DO NOT DESTROY D !
0066 7A                     MOV     A,D
0067 FE02                   CPI     NMBCNS
0069 DA6F00                 JC      TBLJMP
006C F1                     POP     PSW         ; THROW AWAY TABLE ADDRESS
006D AF                     XRA     A
006E C9                     RET
                    TBLJMP:                             ; COMPUTE AND JUMP TO HANDLE
                                                ; A = CONSOLE #
                                                ; DO NOT DESTROY D !
006F 87                     ADD     A           ; DOUBLE CONSOLE # FOR ADR OFFST
0070 E1                     POP     H           ; RETURN ADR POINTS TO JUMP TBL
0071 85                     ADD     L
0072 6F                     MOV     L,A         ; ADD CONSOLE # * 2 TO TBL BASE
```

```
0073 3E00              MVI     A,0
0075 8C                ADC     H
0076 67                MOV     H,A
0077 7E                MOV     A,M        ; GET HANDLER ADDRESS
0078 23                INX     H
0079 66                MOV     H,M
007A 6F                MOV     L,A
007B E9                PCHL               ; JUMP TO COMPUTED CNS HANDLER

                ;
                ; ASCII CHARACTER EQUATES
                ;
005F =          ULINE  EQU     5FH
007F =          RUBOUT EQU     7FH
0020 =          SPACE  EQU     20H
0008 =          BACKSP EQU     8H
005F =          ALTRUB EQU     ULINE
                ;
                ; INPUT / OUTPUT PORT ADDRESS EQUATES
                ;
0040 =          DATA0  EQU     40H
0041 =          STS0   EQU     DATA0+1
0048 =          DATA1  EQU     48H
0049 =          STS1   EQU     DATA1+1
0050 =          DATA2  EQU     50H
0051 =          STS2   EQU     DATA2+1
                ;
                ; POLL CONSOLE #0 INPUT
                ;
                POLCI0:
                PT0ST:                    ; RETURN 0FFH IF READY,
                                          ;        000H IF NOT
007C DB41              IN      STS0
007E E602              ANI     2
0080 C8                RZ
0081 3EFF              MVI     A,0FFH
0083 C9                RET
                ;
                ; CONSOLE #0 INPUT
                ;
                PT0IN:                    ; RETURN CHARACTER IN REG A
0084 C5                PUSH    B
0085 D5                PUSH    D
0086 E5                PUSH    H
0087 0E83              MVI     C,POLL
0089 1E04              MVI     E,PLCI0
008B CDFAFF            CALL    XDOS           ; POLL CONSOLE #0 INPUT
008E E1                POP     H
008F D1                POP     D
0090 C1                POP     B
0091 DB40              IN      DATA0          ; READ CHARACTER
0093 E67F              ANI     7FH            ; STRIP PARITY BIT
0095 C9                RET
                ;
                ; CONSOLE #0 OUTPUT
```

```
                    ;
                    PTØOUT:                                  ; REG C = CHARACTER TO OUTPU
0096 CD9DØØ                 CALL    PTØWAIT                  ; POLL CONSOLE #Ø OUTPUT
0099 79                     MOV     A,C
009A D34Ø                   OUT     DATAØ                    ; TRANSMIT CHARACTER
009C C9                     RET
                    ;
                    ; WAIT FOR CONSOLE #Ø OUTPUT READY
                    ;
                    PTØWAIT:
009D C5                     PUSH    B
009E D5                     PUSH    D
009F E5                     PUSH    H
00A0 ØE83                   MVI     C,POLL
00A2 1E02                   MVI     E,PLCOØ
00A4 CDFAFF                 CALL    XDOS                     ; POLL CONSOLE #Ø OUTPUT
00A7 E1                     POP     H
00A8 D1                     POP     D
00A9 C1                     POP     B
00AA C9                     RET

                    ;
                    ; POLL CONSOLE #Ø OUTPUT
                    ;
                    POLCOØ:
                                                    ; RETURN ØFFH IF READY,
                                                    ;        ØØØH IF NOT
00AB DB41                   IN      STSØ
00AD E6Ø1                   ANI     Ø1H
00AF C8                     RZ
00BØ 3EFF                   MVI     A,ØFFH
00B2 C9                     RET
                    ;
                    ;
                    ; LINE PRINTER DRIVER:  TI 81Ø SERIAL PRINTER
                    ;                       TTY MODEL 4Ø
                    ;
                    INITFLAG:
00B3 ØØ                     DB      Ø       ; PRINTER INITIALIZATION FLAG

                    LIST:                   ; LIST OUTPUT
                    PT1OUT:
                                            ; REG C = CHARACTER TO PRINT
00B4 3AB3ØØ                 LDA     INITFLAG
00B7 B7                     ORA     A
00B8 C2C2ØØ                 JNZ     PT1XX
00BB 3E27                   MVI     A,27H
00BD D349                   OUT     49H                      ; TTY MODEL 4Ø INIT
00BF 32B3ØØ                 STA     INITFLAG
                    PT1XX:
00C2 C5                     PUSH    B
00C3 D5                     PUSH    D
00C4 ØE83                   MVI     C,POLL
00C6 1EØØ                   MVI     E,PLLPT
00C8 CDFAFF                 CALL    XDOS                     ; POLL PRINTER OUTPUT
```

```
00CB D1                 POP     D
00CC C1                 POP     B
00CD 79                 MOV     A,C             ; CHAR TO REGISTER A
00CE D348               OUT     DATA1
00D0 C9                 RET
                ;
                ; POLL PRINTER OUTPUT
                ;
                POLLPT:
                                        ; RETURN 0FFH IF READY,
                                        ;        000H IF NOT
00D1 DB49               IN      STS1
00D3 E601               ANI     01H
00D5 C8                 RZ
00D6 3EFF               MVI     A,0FFH
00D8 C9                 RET
                ;
                ; POLL CONSOLE #1 (PORT 2) INPUT
                ;
                POLCI2:
                PT2ST:
                                        ; RETURN 0FFH IF READY,
                                        ;        000H IF NOT
00D9 DB51               IN      STS2
00DB E602               ANI     2
00DD C8                 RZ
00DE 3EFF               MVI     A,0FFH
00E0 C9                 RET
                ;
                ; CONSOLE #1 (PORT 2) INPUT
                ;
                PT2IN:
                                        ; RETURN CHARACTER IN REG A
00E1 C5                 PUSH    B
00E2 D5                 PUSH    D
00E3 E5                 PUSH    H
00E4 0E83               MVI     C,POLL
00E6 1E05               MVI     E,PLCI2
00E8 CDFAFF             CALL    XDOS            ; POLL CONSOLE #1 INPUT
00EB E1                 POP     H
00EC D1                 POP     D
00ED C1                 POP     B
00EE DB50               IN      DATA2           ; READ CHARACTER
00F0 E67F               ANI     7FH             ; STRIP PARITY BIT
00F2 C9                 RET
                ;
                ; CONSOLE #1 (PORT 2) OUTPUT
                ;
                PT2OUT:
                                        ; REG C = CHARACTER TO OUTPUT
00F3 CD0B01             CALL    PT2WAIT
00F6 79                 MOV     A,C
00F7 D350               OUT     DATA2           ; TRANSMIT CHARACTER
00F9 FE0A               CPI     0AH             ; LINE FEED REQUIRES A DELAY
00FB C0                 RNZ
```

```
00FC C5                 PUSH    B
00FD D5                 PUSH    D
00FE E5                 PUSH    H
00FF 0E8D               MVI     C,XDELAY
0101 110400             LXI     D,4                 ; AT LEAST 3 TICKS = 48 MS
0104 CDFAFF             CALL    XDOS                ; DELAY
0107 E1                 POP     H
0108 D1                 POP     D
0109 C1                 POP     B
010A C9                 RET
                ;
                ; WAIT FOR CONSOLE #1 (PORT 2) OUTPUT READY
                ;
                PT2WAIT:
010B C5                 PUSH    B
010C D5                 PUSH    D
010D E5                 PUSH    H
010E 0E83               MVI     C,POLL
0110 1E03               MVI     E,PLCO2
0112 CDFAFF             CALL    XDOS                ; POLL CONSOLE #1 OUTPUT
0115 E1                 POP     H
0116 D1                 POP     D
0117 C1                 POP     B
0118 C9                 RET
                ;
                ; POLL CONSOLE #1 (PORT 2) OUTPUT
                ;
                POLCO2:
                                        ; RETURN 0FFH IF READY,
                                        ;        000H IF NOT
0119 DB51               IN      STS2
011B E601               ANI     01H
011D C8                 RZ
011E 3EFF               MVI     A,0FFH
0120 C9                 RET
                ;
                ;
                ;  MP/M 1.0   XIOS
                ;
                ;
0006 =          NMBDEV  EQU     6       ; NUMBER OF DEVICES IN POLL TBL

                POLLDEVICE:
                                        ; REG C = DEVICE # TO BE POLLED
                                        ; RETURN 0FFH IF READY,
                                        ;        000H IF NOT
0121 79                 MOV     A,C
0122 FE06               CPI     NMBDEV
0124 DA2901             JC      X010
0127 3E06               MVI     A,NMBDEV; IF DEV # >= NMBDEV,
                                        ; SET TO NMBDEV
                X010:
0129 CD6F00             CALL    TBLJMP  ; JUMP TO DEV POLL CODE

012C D100               DW      POLLPT  ; POLL PRINTER OUTPUT
```

```
012E 9302               DW      POLDSK  ; POLL DISK READY
0130 AB00               DW      POLCO0  ; POLL CONSOLE #0 OUTPUT
0132 1901               DW      POLCO2  ; POLL CONSOLE #1 (PORT 2) OUTPUT
0134 7C00               DW      POLCI0  ; POLL CONSOLE #0 INPUT
0136 D900               DW      POLCI2  ; POLL CONSOLE #1 (PORT 2) INPUT
0138 3A01               DW      BADDEV  ; BAD DEVICE HANDLER
                ;
                BADDEV:                         ; BAD DEVICE NUMBER
                                        ; RETURNS 000H, NOT READY
013A AF                 XRA     A
013B C9                 RET

                ;
                ; SELECT / PROTECT MEMORY
                ;
                SELMEMORY:
                                        ; REG BC = ADR OF MEM DESCRIPTOR
                                        ; *** NOT IMPLEMENTED ***
                                        ; DUMMY RETURN
013C C9                 RET
                ;
                ; START CLOCK
                ;
                STARTCLOCK:
                                        ; WILL CAUSE FLAG #1 TO BE SET
                                        ;  AT EACH SYSTEM TIME UNIT TICK
013D 3EFF               MVI     A,0FFH
013F 323403             STA     TICKN
0142 C9                 RET
                ;
                ; STOP CLOCK
                ;
                STOPCLOCK:
                                        ; WILL STOP FLAG #1 SETTING AT
                                        ;  SYSTEM TIME UNIT TICK
0143 AF                 XRA     A
0144 323403             STA     TICKN
0147 C9                 RET
                ;
                ; EXIT REGION
                ;
                EXITREGION:
                                        ; EI IF NOT PREEMPTED
0148 3A3603             LDA     PREEMP
014B B7                 ORA     A
014C C0                 RNZ
014D FB                 EI
014E C9                 RET
                ;
                ; MAXIMUM CONSOLE NUMBER
                ;
                MAXCONSOLE:
014F 3E02               MVI     A,NMBCNS
0151 C9                 RET
                ;
                ; SYSTEM INITIALIZATION
                ;
                SYSTEMINIT:
                ;
                ;  THIS IS THE PLACE TO INSERT CODE TO INITIALIZE
                ;  THE TIME OF DAY CLOCK, IF IT IS DESIRED ON EACH
                ;  BOOTING OF THE SYSTEM.
                ;
0152 3EC3              MVI     A,0C3H
0154 323800            STA     0038H
0157 216501            LXI     H,INTHND
015A 223900            SHLD    0039H             ; JMP INTHND AT 0038H

015D 3A3503            LDA     INTMSK
0160 D360              OUT     60H               ; INIT INTERRUPT MASK

0162 ED56              DB      0EDH,056H         ; INTERRUPT MODE 1
                                                 ; ** Z80 INSTRUCTION **
0164 C9                RET

                ;
                ;  MP/M 1.0   INTERRUPT HANDLERS
                ;

0085 =          FLAGSET        EQU     133
008E =          DSPTCH EQU     142

                INTHND:
                                             ; INTERRUPT HANDLER ENTRY POINT
                                             ;  ALL INTERRUPTS GEN A RST 7
                                             ;  LOCATION 0038H CONTAINS A JMP
                                             ;  TO INTHND.
0165 F5                PUSH    PSW
0166 223003            SHLD    SVDHL
0169 210000            LXI     H,0
016C 39                DAD     SP
016D 223203            SHLD    SVDSP             ; SAVE USERS STK PTR
0170 313003            LXI     SP,INTSTK+48      ; LCL STK FOR INTR HNDL
0173 D5                PUSH    D
0174 C5                PUSH    B

0175 3EFF              MVI     A,0FFH
0177 323603            STA     PREEMP  ; SET PREEMPTED FLAG

017A DB60              IN      60H               ; READ INTERRUPT MASK
017C E640              ANI     01000000B         ; TEST & JUMP IF CLK INT
017E C28401            JNZ     CLK60HZ
                ;
                ;      ...                       ; TEST/HANDLE OTHER INTS
                ;
0181 C3AA01            JMP     INTDONE

                CLK60HZ:
                                                 ; 60 HZ CLOCK INTERRUPT
0184 3A3403            LDA     TICKN
```

```
0187 B7                 ORA     A               ; TEST TICKN. INDICATES
                                                ;  DELAYED PROCESS(ES)
0188 CA9201             JZ      NOTICKN
018B 0E85               MVI     C,FLAGSET
018D 1E01               MVI     E,1
018F CDFAFF             CALL    XDOS            ; SET FLAG #1 EACH TICK
                NOTICKN:
0192 21FF02             LXI     H,CNT60
0195 35                 DCR     M               ; DEC 60 TICK CNTR
0196 C2A201             JNZ     NOT1SEC
0199 363C               MVI     M,60
019B 0E85               MVI     C,FLAGSET
019D 1E02               MVI     E,2
019F CDFAFF             CALL    XDOS            ; SET FLAG #2 @ 1 SEC
                NOT1SEC:
01A2 AF                 XRA     A
01A3 D360               OUT     60H
01A5 3A3503             LDA     INTMSK
01A8 D360               OUT     60H             ; ACK CLOCK INTERRUPT
                ;       JMP     INTDONE
                ;
                ;
                ;       ...
                ; OTHER INTERRUPT HANDLERS
                ;       ...
                ;
                INTDONE:
01AA AF                 XRA     A
01AB 323603             STA     PREEMP  ; CLEAR PREEMPTED FLAG
01AE C1                 POP     B
01AF D1                 POP     D
01B0 2A3203             LHLD    SVDSP
01B3 F9                 SPHL                    ; RESTORE STK PTR
01B4 2A3003             LHLD    SVDHL
01B7 F1                 POP     PSW
01B8 C3FDFF             JMP     PDISP           ; MP/M DISPATCH
                ;
                ;
                ;       DISK I/O DRIVERS
                ;
                ; DISK PORT EQUATES
                ;
0080 =          CMD1    EQU     80H
0080 =          STAT    EQU     80H
0081 =          HADDR   EQU     81H
0082 =          LADDR   EQU     82H
0083 =          CMD2    EQU     83H
                ;
                ;
                HOME:   ;MOVE TO THE TRACK 00 POSITION OF CURRENT DRIVE
01BB CDDC02             CALL    HEADLOAD
                ; H.L POINT TO WORD WITH TRACK FOR SELECTED DISK
                HOMEL:
01BE 3600               MVI     M,00    ;SET CURRENT TRACK PTR BACK TO 0
01C0 DB80               IN      STAT    ;READ FDC STATUS
01C2 E604               ANI     4       ;TEST TRACK 0 BIT
```

```
01C4 C8                 RZ                  ;RETURN IF AT Ø
01C5 37                 STC                 ;DIRECTION=OUT
01C6 CDC302             CALL    STEP        ;STEP ONE TRACK
01C9 C3BEØ1             JMP     HOMEL       ;LOOP
               ;
               SELDSK:
                        ;DRIVE NUMBER IN C
01CC 21ØØØØ             LXI     H,Ø         ;ØØØØ IN HL PRODUCES SELECT ERROR
01CF 79                 MOV     A,C         ;A IS DISK NUMBER Ø ... NDISKS-1
01DØ FEØ2               CPI     NDISKS      ;LESS THAN NDISKS?
01D2 DØ                 RNC                 ;RETURN WITH HL = ØØØØ IF NOT
               ;MAKE SURE DUMMY IS Ø (FOR USE IN DOUBLE ADD TO H,L)
01D3 AF                 XRA     A
01D4 323FØ3             STA     DUMMY
01D7 79                 MOV     A,C
01D8 E6Ø7               ANI     Ø7H         ;GET ONLY DISK SELECT BITS
01DA 323EØ3             STA     DISKNO
01DD 4F                 MOV     C,A
               ;SET UP THE SECOND COMMAND PORT
01DE 3A41Ø3             LDA     PORT
01E1 E6FØ               ANI     ØFØH        ;CLEAR OUT OLD DISK SELECT BITS
01E3 B1                 ORA     C           ;PUT IN NEW DISK SELECT BITS
01E4 F6Ø8               ORI     Ø8H         ; FORCE DOUBLE DENSITY
01E6 32410З             STA     PORT
               ;        PROPER DISK NUMBER. RETURN DPB ELEMENT ADDRESS
01E9 69                 MOV     L,C
01EA 29                 DAD     H           ;*2
01EB 29                 DAD     H           ;*4
01EC 29                 DAD     H           ;*8
01ED 29                 DAD     H           ;*16
01EE 1144Ø3             LXI     D.DPBASE
01F1 19                 DAD     D           ;HL=.DPB
01F2 2273Ø3             SHLD    TRAN        ;TRANSLATE TABLE BASE
01F5 C9                 RET
               ;
               ;
               ;
               SETTRK:                      ;SET TRACK GIVEN BY REGISTER C
01F6 CDDCØ2             CALL    HEADLOAD
               ;H,L REFERENCE CORRECT TRACK INDICATOR ACCORDING TO
               ;SELECTED DISK
01F9 79                 MOV     A,C         ;DESIRED TRACK
01FA BE                 CMP     M
01FB C8                 RZ                  ;WE ARE ALREADY ON THE TRACK
               SETTKX:
01FC CDC3Ø2             CALL    STEP        ;STEP TRACK-CARRY HAS DIRECTION
                                            ;STEP WILL UPDATE TRK INDICATOR
01FF 79                 MOV     A,C
02ØØ BE                 CMP     M           ;ARE WE WHERE WE WANT TO BE
02Ø1 C2FCØ1             JNZ     SETTKX      ;NOT YET
               ;HAVE STEPPED ENOUGH
               SEEKRT:
               ;DELAY 1Ø MSEC FOR FINAL STEP TIME AND HEAD SETTLE TIME
               ;THE DELAY ROUTINE DELAYS .5 MILLISECOND
02Ø4 3E14               MVI     A.2ØD
```

```
0206 CD0A02            CALL    DELAY
0209 C9                RET                ;END OF SETTRK ROUTINE
                ;
                DELAY: ;ROUTINE TO DELAY C(A)  .5 MILLISECONDS
020A C5                PUSH    B
                DELAY2:
020B 0E86              MVI     C,086H  ;ADJUST FOR .5 MSEC LOOP DELAY
                                       ;THIS IS THE VALUE FOR OUR IMSAI
                LDXA:
020D 0D                DCR     C
020E C20D02            JNZ     LDXA    ;LOOP 1 MSEC
0211 3D                DCR     A
0212 C20B02            JNZ     DELAY2
0215 C1                POP     B
0216 C9                RET             ;END OF DELAY ROUTINE
                ;
                SETSEC:                ;SET SECTOR GIVEN BY REGISTER C
0217 0C                INR     C
0218 79                MOV     A,C
0219 323B03            STA     SECTOR
021C C9                RET
                ;
                SECTRAN:
                       ;SECTOR NUMBER IN C
                       ;TRANSLATE LOGICAL TO PHYSICAL SECTOR
021D 2A7303            LHLD    TRAN    ;HL=..TRANSLATE
0220 5E                MOV     E,M     ;E=LOW(.TRANSLATE)
0221 23                INX     H
0222 56                MOV     D,M     ;DE=.TRANSLATE
0223 7B                MOV     A,E     ;ZERO?
0224 B2                ORA     D       ;00 OR 00 = 00
0225 2600              MVI     H,0
0227 69                MOV     L,C     ;HL = UNTRANSLATED SECTOR
0228 C8                RZ              ;SKIP IF SO
0229 EB                XCHG
022A 42                MOV     B,D     ;BC=00SS
022B 09                DAD     B       ;HL=.TRANSLATE(SECTOR)
022C 6E                MOV     L,M
022D 62                MOV     H,D     ;HL=TRANSLATE(SECTOR)
022E C9                RET
                ;
                SETDMA:                ;SET DMA ADDRESS GIVEN BY REGISTERS B AND C
022F 69                MOV     L,C     ;LOW ORDER ADDRESS
0230 60                MOV     H,B     ;HIGH ORDER ADDRESS
0231 223C03            SHLD    DMAAD   ;SAVE THE ADDRESS
0234 C9                RET
                ;
                ;
                READ:  ;PERFORM READ OPERATION.
                       ;THIS IS SIMILAR TO WRITE, SO SET UP READ
                       ; COMMAND AND USE COMMON CODE IN WRITE
0235 0640              MVI     B,040H  ;SET READ FLAG
0237 C33C02            JMP     WAITIO  ;TO PERFORM THE ACTUAL I/O
                ;
                WRITE: ;PERFORM A WRITE OPERATION
```

```
023A 0680           MVI     B,080H  ;SET WRITE COMMAND
                ;
                WAITIO:
                ;ENTER HERE FROM READ AND WRITE TO PERFORM THE ACTUAL
                ; I/O  OPERATION.  RETURN A 00H IN REGISTER A IF THE
                ; OPERATION COMPLETES PROPERLY, AND 01H IF AN ERROR
                ; OCCURS DURING THE READ OR WRITE
                ;
                ;IN THIS CASE, THE DISK NUMBER SAVED IN 'DISKNO'
                ;                       THE TRACK NUMBER IN 'TRACK'
                ;                       THE SECTOR NUMBER IN 'SECTOR'
                ;                       THE DMA ADDRESS IN 'DMAAD'
                                        ;B STILL HAS R/W FLAG
023C 3E0A           MVI     A,10D   ;SET ERROR COUNT
023E 324003         STA     ERRORS  ;RETRY SOME FAILURES 10 TIMES
                                    ;BEFORE GIVING UP
                TRYAGN:
0241 C5             PUSH    B
0242 CDDC02         CALL    HEADLOAD
                ;H,L POINT TO TRACK BYTE FOR SELECTED DISK
0245 C1             POP     B
0246 4E             MOV     C,M
                ; DECIDE WHETHER TO ALLOW DISK WRITE PRECOMPENSTATION
0247 3E27           MVI     A,39D   ;INHIBIT PRECOMP ON TRKS 0-39
0249 B9             CMP     C
024A DA5102         JC      ALLOWIT
                ;INHIBIT PRECOMP
024D 3E10           MVI     A,10H
024F B0             ORA     B
0250 47             MOV     B,A     ;GOES OUT ON THE SAME PORT
                                    ; AS READ/WRITE
                ALLOWIT:
0251 2A3C03         LHLD    DMAAD   ;GET BUFFER ADDRESS
0254 C5             PUSH    B       ;B HAS R/W CODE    C HAS TRACK
0255 2B             DCX     H       ;SAVE AND REPLACE 3 BYTES BELOW
                                    ;BUF WITH TRK,SCTR.ADR MARK
0256 5E             MOV     E,M
                ;FIGURE CORRECT ADDRESS MARK

0257 3A4103         LDA     PORT
025A E608           ANI     08H
025C 3EFB           MVI     A.0FBH
025E CA6302         JZ      SIN
0261 E60F           ANI     0FH     ;WAS DOUBLE
                                    ;0BH IS DOUBLE DENSITY
                                    ;0FBH IS SINGLE DENSITY
                SIN:
0263 77             MOV     M,A
                ;FILL IN SECTOR
0264 2B             DCX     H
0265 56             MOV     D.M
0266 3A3B03         LDA     SECTOR  ;NOTE THAT INVALID SECTOR NUMBER
                                    ;WILL RESULT IN HEAD UNLOADED
                                    ;ERROR, SO DONT CHECK
0269 77             MOV     M,A
```

```
                   ;FILL IN TRACK
026A 2B                   DCX     H
026B C1                   POP     B
026C 79                   MOV     A,C
026D 4E                   MOV     C,M
026E 77                   MOV     M,A
026F 7C                   MOV     A,H       ;SET UP FDC DMA ADDRESS
0270 D381                 OUT     HADDR     ;HIGH BYTE
0272 7D                   MOV     A,L
0273 D382                 OUT     LADDR     ;LOW BYTE
0275 78                   MOV     A,B       ;GET R/W FLAG
0276 D380                 OUT     CMD1      ;START DISK READ/WRITE

                   RWWAIT:
0278 C5                   PUSH    B
0279 D5                   PUSH    D
027A E5                   PUSH    H
027B 0E83                 MVI     C,POLL
027D 1E01                 MVI     E,PLDSK
027F CDFAFF               CALL    XDOS              ; POLL DISK READY
0282 E1                   POP     H
0283 D1                   POP     D
0284 C1                   POP     B
0285 AF                   XRA     A

0286 71                   MOV     M,C       ;RESTORE 3 BYTES BELOW BUF
0287 23                   INX     H
0288 72                   MOV     M,D
0289 23                   INX     H
028A 73                   MOV     M,E
028B DB80                 IN      STAT      ;TEST FOR ERRORS
028D E6F0                 ANI     0F0H
028F C8                   RZ                ;A WILL BE 0 IF NO ERRORS
0290 C39B02               JMP     ERRTN
                   ;
                   ; POLL DISK READY
                   ;
                   POLDSK:
                                            ; RETURN 0FFH IF READY,
                                            ;        000H IF NOT
0293 DB80                 IN      STAT      ; READ FDC STATUS
0295 E688                 ANI     88H       ; TEST FOR HEAD UNLOAD OR IOF
0297 C8                   RZ
0298 3EFF                 MVI     A,0FFH
029A C9                   RET

                   ERRTN:
                   ;COME HERE ON ERROR FROM DISK
029B F5                   PUSH    PSW       ;SAVE ERROR CONDITION
                   ;CHECK FOR 10 ERRORS
029C 214003               LXI     H,ERRORS
029F 35                   DCR     M
02A0 C2A702               JNZ     REDO      ;NOT TEN YET.  DO A RETRY
                   ;WE HAVE TOO MANY ERRORS. PRINT OUT HEX NUMBER FOR LAST
                   ;RECEIVED ERROR TYPE. CPM WILL PRINT PERM ERROR MESSAGE.
```

```
02A3 F1                  POP     PSW      ;GET CODE
                 ;SET ERROR RETURN FOR OPERATING SYSTEM
02A4 3E01                MVI     A,1
02A6 C9                  RET
                 REDO:
                 ;B STILL HAS READ/WRITE FLAG
02A7 F1                  POP     PSW      ;GET ERROR CODE
02A8 E6E0                ANI     0E0H     ;RETRY IF NOT TRACK ERROR
02AA C24102              JNZ     TRYAGN   ;
                 ;WAS A TRACK ERROR SO NEED TO RESEEK
02AD C5                  PUSH    B        ;SAVE   READ/WRITE INDICATOR
                 ;FIGURE OUT THE DESIRED TRACK
02AE 113703              LXI     D,TRACK
02B1 2A3E03              LHLD    DISKNO   ;SELECTED DISK
02B4 19                  DAD     D        ;POINT TO CORRECT TRK INDICATOR
02B5 7E                  MOV     A,M      ;DESIRED TRACK
02B6 F5                  PUSH    PSW      ;SAVE IT
02B7 CDBB01              CALL    HOME
02BA F1                  POP     PSW
02BB 4F                  MOV     C,A
02BC CDF601              CALL    SETTRK
02BF C1                  POP     B        ;GET READ/WRITE INDICATOR
02C0 C34102              JMP     TRYAGN
                 ;
                 ;
                 ;
                 STEP:                    ;STEP HEAD OUT TOWARDS ZERO
                                          ;IF CARRY IS SET; ELSE
                                          ;STEP IN
                 ; H,L POINT TO CORRECT TRACK INDICATOR WORD
02C3 DAD702              JC      OUTX
02C6 34                  INR     M        ;INCREMENT CURRENT TRACK BYTE
02C7 3E04                MVI     A,04H    ;SET DIRECTION = IN
                 DOSTEP:
02C9 F602                ORI     2
02CB D380                OUT     CMD1     ;PULSE STEP BIT
02CD E6FD                ANI     0FDH
02CF D380                OUT     CMD1     ;TURN OFF PULSE
                 ;THE FDC-2 HAD A STEPP READY LINE. THE FDC-3 RELIES ON
                 ;SOFTWARE TIME OUT
02D1 3E10                MVI     A,16D    ;WAIT FOR STEP READY
                 ;DELAY ROUTINE DELAYS FOR .5 MSEC TIMES THE
                 ; CONTENTS OF REG A
02D3 CD0A02              CALL    DELAY
02D6 C9                  RET
                 ;
                 OUTX:
02D7 35                  DCR     M        ;UPDATE TRACK BYTE
02D8 AF                  XRA     A
02D9 C3C902              JMP     DOSTEP
                 ;
                 HEADLOAD:
                 ;SELECT AND LOAD THE HEAD ON THE CORRECT DRIVE
02DC 214203              LXI     H,PRTOUT         ;OLD SLECT INFO
02DF 46                  MOV     B,M
```

```
02E0 2B                 DCX     H         ;NEW SELECT INFO
02E1 7E                 MOV     A.M
02E2 23                 INX     H
02E3 77                 MOV     M,A
02E4 D383               OUT     CMD2      ;SELECT THE DRIVE
                ;SET UP H.L TO POINT TO TRACK BYTE FOR SELECTED DISK
02E6 113703             LXI     D,TRACK
02E9 2A3E03             LHLD    DISKNO
02EC 19                 DAD     D
                ;NOW CHECK FOR NEEDING A 35 MS DELAY
                ;IF WE HAVE CHANGED DRIVES OR IF THE HEAD IS UNLOADED
                ;WE NEED TO WAIT 35 MS FOR HEAD SETTLE
02ED B8                 CMP     B         ;ARE WE ON THE SAME DRIVE
02EE C2F602             JNZ     NEEDDLY
                ;WE ARE ON THE SAME DRIVE
                ;IS THE HEAD LOADED?
02F1 DB80               IN      STAT
02F3 E680               ANI     80H
02F5 C8                 RZ                ;ALREADY LOADED
                NEEDDLY:
02F6 AF                 XRA     A
02F7 D380               OUT     CMD1      ;LOAD THE HEAD
                ;THE DELAY ROUTINE DELAYS FOR .5 MSEC
02F9 3E46               MVI     A,70D
02FB CD0A02             CALL    DELAY
02FE C9                 RET
                ;
                ;
                ; BIOS DATA SEGMENT
                ;
02FF 3C         CNT60: DB       60        ; 60 TICK CNTR = 1 SEC
0300            INTSTK:         DS        48        ; LOCAL INTRPT STK
0330 0000       SVDHL: DW       0         ; SAVED REGS HL DURING INT HNDL
0332 0000       SVDSP: DW       0         ; SAVED SP DURING INT HNDL
0334 00         TICKN: DB       0         ; TICKING BOOLEAN.TRUE = DELAYED
0335 40         INTMSK:         DB        40H       ; INTRPT MSK, ENABLES CLK IN
0336 00         PREEMP:         DB        0         ; PREEMPTED BOOLEAN
                ;
                SCRAT:                    ; START OF SCRATCH AREA
0337 00         TRACK: DB       0         ; CURRENT TRK ON DRIVE 0
0338 00         TRAK1: DB       0         ; CURRENT TRK ON DRIVE 1
0339 00         TRAK2: DB       0
033A 00         TRAK3: DB       0
033B 00         SECTOR:         DB        0         ; CURRENTLY SELECTED SCTR
033C 0000       DMAAD: DW       0         ; CURRENT DMA ADDRESS
033E 00         DISKNO:         DB        0         ; CURRENT DISK NUMBER
033F 00         DUMMY: DB       0         ; MUST BE 0 FOR DBL ADD
0340 00         ERRORS:         DB        0
0341 00         PORT:  DB       0
0342 00         PRTOUT:         DB        0
0343 00         DNSTY: DB       0
                ;
                        DISKS   2
    +
0002+#          NDISKS SET      2
```

```
0344+=          DPBASE EQU     $          ;BASE OF DISK PARAMETER BLOCKS
0000+#          DSKNXT SET     0
    +                  REPT    2
    +                  DSKHDR  %DSKNXT
    +           DSKNXT SET     DSKNXT+1
    +                  ENDM
    +                  DSKHDR  %DSKNXT
    +
0344+00000000   DPE0:  DW      XLT0,0000H         ;TRANSLATE TABLE
0348+00000000          DW      0000H,0000H        ;SCRATCH AREA
034C+75036403          DW      DIRBUF,DPB0        ;DIR BUFF,PARM BLOCK
0350+1504F503          DW      CSV0,ALV0          ;CHECK, ALLOC VECTORS
    +                  ENDM
0001+#          DSKNXT SET     DSKNXT+1
    +                  DSKHDR  %DSKNXT
    +
0354+00000000   DPE1:  DW      XLT1,0000H         ;TRANSLATE TABLE
0358+00000000          DW      0000H,0000H        ;SCRATCH AREA
035C+75036403          DW      DIRBUF,DPB1        ;DIR BUFF.PARM BLOCK
0360+35041504          DW      CSV1,ALV1          ;CHECK, ALLOC VECTORS
    +                  ENDM
0002+#          DSKNXT SET     DSKNXT+1
    +                  ENDM
    +                  ENDM
0800 =          BPB    EQU     2*1024   ;BYTES PER BLOCK
0010 =          RPB    EQU     BPB/128 ;RECORDS PER BLOCK
00FF =          MAXB   EQU     255      ;MAX BLOCK NUMBER
                       DISKDEF 0,1,58,,BPB,MAXB+1,128,0,2
    +
    +                  IF      NUL 58
    +           DPB0   EQU     DPB1     ;EQUIVALENT PARAMETERS
    +           ALS0   EQU     ALS1     ;SAME ALLOCATION VECTOR SIZE
    +           CSS0   EQU     CSS1     ;SAME CHECKSUM VECTOR SIZE
    +           XLT0   EQU     XLT1     ;SAME TRANSLATE TABLE
    +                  ELSE
0039+#          SECMAX SET     58-(1)
003A+#          SECTORS        SET      SECMAX+1
0020+#          ALS0   SET     (MAXB+1)/8
    +                  IF      ((MAXB+1) MOD 8) NE 0
    +           ALS0   SET     ALS0+1
    +                  ENDIF
0000+#          CSS0   SET     (0)/4
0010+#          BLKVAL SET     BPB/128
0000+#          BLKSHF SET     0
0000+#          BLKMSK SET     0
    +                  REPT    16
    +                  IF      BLKVAL=1
    +                  EXITM
    +                  ENDIF
    +           BLKSHF SET     BLKSHF+1
    +           BLKMSK SET     (BLKMSK SHL 1) OR 1
    +           BLKVAL SET     BLKVAL/2
    +                  ENDM
    +                  IF      BLKVAL=1
    +                  EXITM
    +                  ENDIF
0001+#          BLKSHF SET     BLKSHF+1
0001+#          BLKMSK SET     (BLKMSK SHL 1) OR 1
0008+#          BLKVAL SET     BLKVAL/2
    +                  IF      BLKVAL=1
    +                  EXITM
    +                  ENDIF
0002+#          BLKSHF SET     BLKSHF+1
0003+#          BLKMSK SET     (BLKMSK SHL 1) OR 1
0004+#          BLKVAL SET     BLKVAL/2
    +                  IF      BLKVAL=1
    +                  EXITM
    +                  ENDIF
0003+#          BLKSHF SET     BLKSHF+1
0007+#          BLKMSK SET     (BLKMSK SHL 1) OR 1
0002+#          BLKVAL SET     BLKVAL/2
    +                  IF      BLKVAL=1
    +                  EXITM
    +                  ENDIF
0004+#          BLKSHF SET     BLKSHF+1
000F+#          BLKMSK SET     (BLKMSK SHL 1) OR 1
0001+#          BLKVAL SET     BLKVAL/2
    +                  IF      BLKVAL=1
    +                  EXITM
0002+#          BLKVAL SET     BPB/1024
0000+#          EXTMSK SET     0
    +                  REPT    16
    +                  IF      BLKVAL=1
    +                  EXITM
    +                  ENDIF
    +           EXTMSK SET     (EXTMSK SHL 1) OR 1
    +           BLKVAL SET     BLKVAL/2
    +                  ENDM
    +                  IF      BLKVAL=1
    +                  EXITM
    +                  ENDIF
0001+#          EXTMSK SET     (EXTMSK SHL 1) OR 1
0001+#          BLKVAL SET     BLKVAL/2
    +                  IF      BLKVAL=1
    +                  EXITM
    +                  IF      (MAXB+1) > 256
    +           EXTMSK SET     (EXTMSK SHR 1)
    +                  ENDIF
    +                  IF      NOT NUL
    +           EXTMSK SET
    +                  ENDIF
0080+#          DIRREM SET     128
0040+#          DIRBKS SET     BPB/32
0000+#          DIRBLK SET     0
    +                  REPT    16
    +                  IF      DIRREM=0
    +                  EXITM
    +                  ENDIF
    +           DIRBLK SET     (DIRBLK SHR 1) OR 8000H
    +                  IF      DIRREM > DIRBKS
    +          DIRREM SET     DIRREM-DIRBKS
    +                 ELSE
    +          DIRREM SET     Ø
    +                 ENDIF
    +                 ENDM
    +                 IF      DIRREM=Ø
    +                 EXITM
    +                 ENDIF
8000+#         DIRBLK SET     (DIRBLK SHR 1) OR 8ØØØH
    +                 IF      DIRREM > DIRBKS
ØØ4Ø+#         DIRREM SET     DIRREM-DIRBKS
    +                 ELSE
    +          DIRREM SET     Ø
    +                 ENDIF
    +                 IF      DIRREM=Ø
    +                 EXITM
    +                 ENDIF
CØØØ+#         DIRBLK SET     (DIRBLK SHR 1) OR 8ØØØH
    +                 IF      DIRREM > DIRBKS
    +          DIRREM SET     DIRREM-DIRBKS
    +                 ELSE
ØØØØ+#         DIRREM SET     Ø
    +                 ENDIF
    +                 IF      DIRREM=Ø
    +                 EXITM
    +                 DPBHDR  Ø
Ø364+=         DPBØ   EQU     $                 ;DISK PARM BLOCK
    +                 ENDM
    +                 DDW     %SECTORS,<;SEC PER TRACK>
    +
Ø364+3AØØ             DW      58                ;SEC PER TRACK
    +                 ENDM
    +                 DDB     %BLKSHF,<;BLOCK SHIFT>
    +
Ø366+Ø4               DB      4                 ;BLOCK SHIFT
    +                 ENDM
    +                 DDB     %BLKMSK,<;BLOCK MASK>
    +
Ø367+ØF               DB      15                ;BLOCK MASK
    +                 ENDM
    +                 DDB     %EXTMSK,<;EXTNT MASK>
    +
Ø368+Ø1               DB      1                 ;EXTNT MASK
    +                 ENDM
    +                 DDW     %(MAXB+1)-1,<;DISK SIZE-1>
    +
Ø369+FFØØ             DW      255               ;DISK SIZE-1
    +                 ENDM
    +                 DDW     %(128)-1,<;DIRECTORY MAX>
    +
Ø36B+7FØØ             DW      127               ;DIRECTORY MAX
    +                 ENDM
    +                 DDB     %DIRBLK SHR 8,<;ALLOCØ>
    +
Ø36D+CØ               DB      192               ;ALLOCØ
```

```
     +                  ENDM
     +                  DDB     %DIRBLK AND 0FFH,<;ALLOC1>
     +
036E+00                 DB      0                ;ALLOC1
     +                  ENDM
     +                  DDW     %(0)/4,<;CHECK SIZE>
     +
036F+0000               DW      0                ;CHECK SIZE
     +                  ENDM
     +                  DDW     %2,<;OFFSET>
     +
0371+0200               DW      2                ;OFFSET
     +                  ENDM
     +                  IF      NUL
0000+=          XLT0    EQU     0                ;NO XLATE TABLE
     +                  ELSE
     +                  IF       = 0
     +          XLT0    EQU     0                ;NO XLATE TABLE
     +                  ELSE
     +          NXTSEC  SET     0
     +          NXTBAS  SET     0
     +                  GCD     %SECTORS,
     +          NELTST  SET     SECTORS/GCDN
     +          NELTS   SET     NELTST
     +          XLT0    EQU     $                ;TRANSLATE TABLE
     +                  REPT    SECTORS
     +                  IF      SECTORS < 256
     +                  DDB     %NXTSEC+(1)
     +                  ELSE
     +                  DDW     %NXTSEC+(1)
     +                  ENDIF
     +          NXTSEC  SET     NXTSEC+()
     +                  IF      NXTSEC >= SECTORS
     +          NXTSEC  SET     NXTSEC-SECTORS
     +                  ENDIF
     +          NELTS   SET     NELTS-1
     +                  IF      NELTS = 0
     +          NXTBAS  SET     NXTBAS+1
     +          NXTSEC  SET     NXTBAS
     +          NELTS   SET     NELTST
     +                  ENDIF
     +                  ENDM
     +                  ENDIF
     +                  ENDIF
     +                  ENDM
                        DISKDEF 1,0
     +
     +                  IF      NUL
0364+=          DPB1    EQU     DPB0    ;EQUIVALENT PARAMETERS
0020+=          ALS1    EQU     ALS0    ;SAME ALLOCATION VECTOR SIZE
0000+=          CSS1    EQU     CSS0    ;SAME CHECKSUM VECTOR SIZE
0000+=          XLT1    EQU     XLT0    ;SAME TRANSLATE TABLE
     +                  ELSE
     +          SECMAX  SET     -(0)
     +          SECTORS         SET     SECMAX+1
     +          ALS1   SET      ()/8
     +                 IF       (() MOD 8) NE Ø
     +          ALS1   SET      ALS1+1
     +                 ENDIF
     +          CSS1   SET      ()/4
     +          BLKVAL SET      /128
     +          BLKSHF SET      Ø
     +          BLKMSK SET      Ø
     +                 REPT     16
     +                 IF       BLKVAL=1
     +                 EXITM
     +                 ENDIF
     +          BLKSHF SET      BLKSHF+1
     +          BLKMSK SET      (BLKMSK SHL 1) OR 1
     +          BLKVAL SET      BLKVAL/2
     +                 ENDM
     +          BLKVAL SET      /1Ø24
     +          EXTMSK SET      Ø
     +                 REPT     16
     +                 IF       BLKVAL=1
     +                 EXITM
     +                 ENDIF
     +          EXTMSK SET      (EXTMSK SHL 1) OR 1
     +          BLKVAL SET      BLKVAL/2
     +                 ENDM
     +                 IF       () > 256
     +          EXTMSK SET      (EXTMSK SHR 1)
     +                 ENDIF
     +                 IF       NOT NUL
     +          EXTMSK SET
     +                 ENDIF
     +          DIRREM SET
     +          DIRBKS SET      /32
     +          DIRBLK SET      Ø
     +                 REPT     16
     +                 IF       DIRREM=Ø
     +                 EXITM
     +                 ENDIF
     +          DIRBLK SET      (DIRBLK SHR 1) OR 8ØØØH
     +                 IF       DIRREM > DIRBKS
     +          DIRREM SET      DIRREM-DIRBKS
     +                 ELSE
     +          DIRREM SET      Ø
     +                 ENDIF
     +                 ENDM
     +                 DPBHDR   1
     +                 DDW      %SECTORS,<;SEC PER TRACK>
     +                 DDB      %BLKSHF,<;BLOCK SHIFT>
     +                 DDB      %BLKMSK,<;BLOCK MASK>
     +                 DDB      %EXTMSK,<;EXTNT MASK>
     +                 DDW      %()-1,<;DISK SIZE-1>
     +                 DDW      %()-1,<;DIRECTORY MAX>
     +                 DDB      %DIRBLK SHR 8,<;ALLOCØ>
     +                 DDB      %DIRBLK AND ØFFH,<;ALLOC1>
     +                 DDW      %()/4,<;CHECK SIZE>
     +                  DDW     %,<;OFFSET>
     +                  IF      NUL
     +           XLT1   EQU     0                ;NO XLATE TABLE
     +                  ELSE
     +                  IF        = 0
     +           XLT1   EQU     0                ;NO XLATE TABLE
     +                  ELSE
     +           NXTSEC SET     0
     +           NXTBAS SET     0
     +                  GCD     %SECTORS,
     +           NELTST SET     SECTORS/GCDN
     +           NELTS  SET     NELTST
     +           XLT1   EQU     $                ;TRANSLATE TABLE
     +                  REPT    SECTORS
     +                  IF      SECTORS < 256
     +                  DDB     %NXTSEC+(0)
     +                  ELSE
     +                  DDW     %NXTSEC+(0)
     +                  ENDIF
     +           NXTSEC SET     NXTSEC+()
     +                  IF      NXTSEC >= SECTORS
     +           NXTSEC SET     NXTSEC-SECTORS
     +                  ENDIF
     +           NELTS  SET     NELTS-1
     +                  IF      NELTS = 0
     +           NXTBAS SET     NXTBAS+1
     +           NXTSEC SET     NXTBAS
     +           NELTS  SET     NELTST
     +                  ENDIF
     +                  ENDM
     +                  ENDIF
     +                  ENDIF
     +                  ENDM
                 ;
0373             TRAN:  DS      2
                 ;
                        ENDEF
     +
0375+=           BEGDAT EQU     $
0375+            DIRBUF:        DS      128      ;DIRECTORY ACCESS BUFFER
0000+#           DSKNXT SET     0
     +                  REPT    NDISKS
     +                  LDS     ALV,%DSKNXT,ALS
     +                  LDS     CSV,%DSKNXT,CSS
     +           DSKNXT SET     DSKNXT+1
     +                  ENDM
     +                  LDS     ALV,%DSKNXT,ALS
     +                  DEFDS   ALV0,%ALS0
03F5+            ALV0:  DS      32
     +                  ENDM
     +                  ENDM
     +                  LDS     CSV,%DSKNXT,CSS
     +                  DEFDS   CSV0,%CSS0
0415+            CSV0:  DS      0
     +                  ENDM
```

```
    +                  ENDM
Ø0Ø1+#        DSKNXT   SET      DSKNXT+1
    +                  LDS      ALV,%DSKNXT,ALS
    +                  DEFDS    ALV1,%ALS1
Ø415+         ALV1:    DS       32
    +                  ENDM
    +                  ENDM
    +                  LDS      CSV,%DSKNXT,CSS
    +                  DEFDS    CSV1,%CSS1
Ø435+         CSV1:    DS       Ø
    +                  ENDM
    +                  ENDM
ØØØ2+#        DSKNXT   SET      DSKNXT+1
    +                  ENDM
Ø435+=        ENDDAT   EQU      $
ØØCØ+=        DATSIZ   EQU      $-BEGDAT
Ø435+ØØ       FORCE:   DB       Ø         ;FORCE OUT LAST BYTE IN HEX FILE
    +                  ENDM

Ø436                   END
Ø251 ALLOWIT   ØØ2Ø ALS1      ØØ5F ALTRUB    Ø3F5 ALVØ      Ø415 ALV1
ØØØ8 BACKSP    Ø13A BADDEV    Ø375 BEGDAT    Ø8ØØ BPB       Ø184 CLK6ØHZ
ØØ8Ø CMD1      ØØ83 CMD2      Ø2FF CNT6Ø     ØØ48 COLDSTART ØØ56 CONIN
ØØ5F CONOUT    ØØ4D CONST     ØØØØ CSS1      Ø415 CSVØ      Ø435 CSV1
ØØ4Ø DATAØ     ØØ48 DATA1     ØØ5Ø DATA2     ØØCØ DATSIZ    Ø2ØA DELAY
Ø2ØB DELAY2    Ø375 DIRBUF    Ø33E DISKNO    Ø33C DMAAD     Ø343 DNSTY
Ø2C9 DOSTEP    Ø364 DPBØ      Ø364 DPB1      Ø344 DPBASE    Ø344 DPEØ
Ø354 DPE1      ØØ8E DSPTCH    Ø33F DUMMY     Ø435 ENDDAT    Ø34Ø ERRORS
Ø29B ERRTN     Ø148 EXITREGION ØØØØ FALSE    ØØ85 FLAGSET   Ø435 FORCE
ØØ81 HADDR     Ø2DC HEADLOAD  Ø1BB HOME      Ø1BE HOMEL     ØØB3 INITF
Ø1AA INTDONE   Ø165 INTHND    Ø335 INTMSK    Ø3ØØ INTSTK    ØØ82 LADDR
Ø2ØD LDXA      ØØB4 LIST      ØØFF MAXB      Ø14F MAXCONSOLE Ø2F6 NEEDDLY
ØØØ2 NMBCNS    ØØØ6 NMBDEV    Ø1A2 NOT1SEC   Ø192 NOTICKN   Ø2D7 OUTX
FFFD PDISP     ØØØ4 PLCIØ     ØØØ5 PLCI2     ØØØ2 PLCOØ     ØØØ3 PLCO2
ØØØ1 PLDSK     ØØØØ PLLPT     ØØ7C POLCIØ    ØØD9 POLCI2    ØØAB POLCOØ
Ø119 POLCO2    Ø293 POLDSK    Ø121 POLLDEVICE ØØ63 POLL     ØØD1 POLLPT
Ø341 PORT      Ø336 PREEMP    Ø342 PRTOUT    ØØ84 PTØIN     ØØ96 PTØOUT
ØØ7C PTØST     ØØ9D PTØWAIT   ØØB4 PT1OUT    ØØC2 PT1XX     ØØE1 PT2IN
ØØF3 PT2OUT    ØØD9 PT2ST     Ø1ØB PT2WAIT   ØØ66 PTBLJMP   ØØ5D PUNCH
ØØ54 READER    Ø235 READ      Ø2A7 REDO      ØØ1Ø RPB       ØØ7F RUBOUT
Ø278 RWWAIT    Ø337 SCRAT     Ø33B SECTOR    Ø21D SECTRAN   Ø2Ø4 SEEKRT
Ø1CC SELDSK    Ø13C SELMEMORY Ø22F SETDMA    Ø217 SETSEC    Ø1FC SETTKX
Ø1F6 SETTRK    Ø263 SIN       ØØ2Ø SPACE     Ø13D STARTCLOCK ØØ8Ø STAT
Ø2C3 STEP      Ø143 STOPCLOCK ØØ41 STSØ      ØØ49 STS1      ØØ51 STS2
Ø33Ø SVDHL     Ø332 SVDSP     Ø152 SYSTEMINIT ØØ6F TBLJMP   Ø334 TICKN
Ø337 TRACK     Ø338 TRAK1     Ø339 TRAK2     Ø33A TRAK3     Ø373 TRAN
FFFF TRUE      Ø241 TRYAGN    ØØ5F ULINE     Ø23C WAITIO    ØØ48 WARMSTA
ØØØ3 WBOOT     Ø23A WRITE     Ø129 XØ1Ø      ØØ8D XDELAY    FFFA XDOS
ØØØØ XLTØ      ØØØØ XLT1
```

Sample Page Relocatable Program

This appendix contains a sample page relocatable program. It illustrates the required use of ORG statements to access the BDOS and the default file control block. Note that the initial ORG is at zero. Its purpose is to establish the equate for BASE, the base of the relocatable segment. Next an ORG 100H statement establishes the actual beginning of code for the program.

It is VERY important to use BASE to offset all relative page zero references! Do not make a call to absolute 0005H for BDOS calls. In this example BASE is used to offset the BDOS, FCB, and BUFF equates. If the user program needed to determine the top of its memory segment the following equate and code sequence should be used:

```
MEMSIZE EQU     BASE+6

...

        LHLD    MEMSIZE ;HL = TOP OF MEMORY SEGMENT
```

The following procedure shows how to generate a page relocatable file for this example:

* Prepare the user program, DUMP.ASM in this example, with proper origin statements as described above.

* Assuming a system disk in drive A: and the DUMP.ASM file is on drive B:, enter the commands-

```
A>MAC B:DUMP $PP+S
      ;ASSEMBLE THE DUMP.ASM FILE, LIST WITH SYMBOL TABLE
A>ERA B:DUMP.HX0
A>REN B:DUMP.HX0=B:DUMP.HEX
A>MAC B:DUMP $PZSZ+R
      ;ASSEMBLE THE DUMP.ASM FILE AGAIN OFFSET BY 100H
      ;THE OFFSET IS GENERATED WITH THE +R MAC OPTION
A>PIP B:DUMP.HEX=B:DUMP.HX0[I],B:DUMP.HEX[H]
      ;CONCATENATE THE HEX FILES
A>GENMOD B:DUMP.HEX B:DUMP.PRL
      ;GENERATE THE RELOCATABLE DUMP.PRL FILE
```

```
                              PAGE    Ø
                              TITLE   'File Dump Program'
                     ;        FILE DUMP PROGRAM, READS AN INPUT FILE AND
                     ;          PRINTS IN HEX
                     ;
                     ;        COPYRIGHT (C) 1975, 1976, 1977, 1978, 1979
                     ;        DIGITAL RESEARCH
                     ;        BOX 579, PACIFIC GROVE
                     ;        CALIFORNIA, 9395Ø
                     ;
ØØØØ                          ORG     ØØØØH    ;BASE OF RELOCATABLE SEGMENT
ØØØØ =               BASE     EQU     $

Ø1ØØ                          ORG     Ø1ØØH    ;BASE OF MP/M PROGRAM AREA
ØØØ5 =               BDOS     EQU     BASE+5   ;DOS ENTRY POINT
ØØØ1 =               CONS     EQU     1        ;READ CONSOLE
ØØØ2 =               TYPEF    EQU     2        ;TYPE FUNCTION
ØØØ9 =               PRINTF   EQU     9        ;BUFFER PRINT ENTRY
ØØØB =               BRKF     EQU     11       ;BREAK KEY FUNCTION
ØØØF =               OPENF    EQU     15       ;FILE OPEN
ØØ14 =               READF    EQU     2Ø       ;READ FUNCTION
                     ;
ØØ5C =               FCB      EQU     BASE+5CH;FILE CONTROL BLOCK ADDRESS
ØØ8Ø =               BUFF     EQU     BASE+8ØH;INPUT DISK BUFFER ADDRESS
                     ;
                     ;        NON GRAPHIC CHARACTERS
ØØØD =               CR       EQU     ØDH      ;CARRIAGE RETURN
ØØØA =               LF       EQU     ØAH      ;LINE FEED
                     ;
                     ;        FILE CONTROL BLOCK DEFINITIONS
ØØ5C =               FCBDN    EQU     FCB+Ø    ;DISK NAME
ØØ5D =               FCBFN    EQU     FCB+1    ;FILE NAME
ØØ65 =               FCBFT    EQU     FCB+9    ;DISK FILE TYPE (3 CHARACTERS)
ØØ68 =               FCBRL    EQU     FCB+12   ;FILE'S CURRENT REEL NUMBER
ØØ6B =               FCBRC    EQU     FCB+15   ;FILE'S RECORD COUNT (Ø TO 128)
ØØ7C =               FCBCR    EQU     FCB+32   ;CURRENT (NEXT) RECORD NUMBER
ØØ7D =               FCBLN    EQU     FCB+33   ;FCB LENGTH
                     ;
                     ;        SET UP STACK
Ø1ØØ 21ØØØØ                   LXI     H,Ø
Ø1Ø3 39                       DAD     SP
                     ;        ENTRY STACK POINTER IN HL FROM THE CCP
Ø1Ø4 221AØ2                   SHLD    OLDSP
                     ;        SET SP TO LOCAL STACK AREA (RESTORED AT FINIS)
Ø1Ø7 315CØ2                   LXI     SP,STKTOP
                     ;        READ AND PRINT SUCCESSIVE BUFFERS
Ø1ØA CDC1Ø1                   CALL    SETUP    ;SET UP INPUT FILE
Ø1ØD FEFF                     CPI     255      ;255 IF FILE NOT PRESENT
Ø1ØF C21BØ1                   JNZ     OPENOK   ;SKIP IF OPEN IS OK
                     ;
                     ;        FILE NOT THERE, GIVE ERROR MESSAGE AND RETURN
Ø112 11F8Ø1                   LXI     D,OPNMSG
Ø115 CD9CØ1                   CALL    ERR
Ø118 C351Ø1                   JMP     FINIS    ;TO RETURN
                     ;
```

```
                    OPENOK:        ;OPEN OPERATION OK, SET BUFFER INDEX TO END
011B 3E80                  MVI     A,80H
011D 321802                STA     IBP      ;SET BUFFER POINTER TO 80H
                    ;      HL CONTAINS NEXT ADDRESS TO PRINT
0120 210000                LXI     H,0      ;START WITH 0000
                    ;
                    GLOOP:
0123 E5                    PUSH    H        ;SAVE LINE POSITION
0124 CDA201                CALL    GNB
0127 E1                    POP     H        ;RECALL LINE POSITION
0128 DA5101                JC      FINIS    ;CARRY SET BY GNB IF END FILE
012B 47                    MOV     B,A
                    ;      PRINT HEX VALUES
                    ;      CHECK FOR LINE FOLD
012C 7D                    MOV     A,L
012D E60F                  ANI     0FH      ;CHECK LOW 4 BITS
012F C24401                JNZ     NONUM
                    ;      PRINT LINE NUMBER
0132 CD7201                CALL    CRLF
                    ;
                    ;      CHECK FOR BREAK KEY
0135 CD5901                CALL    BREAK
                    ;      ACCUM LSB = 1 IF CHARACTER READY
0138 0F                    RRC              ;INTO CARRY
0139 DA5101                JC      FINIS    ;DON'T PRINT ANY MORE
                    ;
013C 7C                    MOV     A,H
013D CD8F01                CALL    PHEX
0140 7D                    MOV     A,L
0141 CD8F01                CALL    PHEX
                    NONUM:
0144 23                    INX     H        ;TO NEXT LINE NUMBER
0145 3E20                  MVI     A,' '
0147 CD6501                CALL    PCHAR
014A 78                    MOV     A,B
014B CD8F01                CALL    PHEX
014E C32301                JMP     GLOOP
                    ;
                    FINIS:
                    ;      END OF DUMP, RETURN TO CCP
                    ;      (NOTE THAT A JMP TO 0000H REBOOTS)
0151 CD7201                CALL    CRLF
0154 2A1A02                LHLD    OLDSP
0157 F9                    SPHL
                    ;      STACK POINTER CONTAINS CCP'S STACK LOCATION
0158 C9                    RET              ;TO THE CCP
                    ;
                    ;
                    ;      SUBROUTINES
                    ;
                    BREAK: ;CHECK BREAK KEY (ACTUALLY ANY KEY WILL DO)
0159 E5D5C5                PUSH H! PUSH D! PUSH B; ENVIRONMENT SAVED
015C 0E0B                  MVI     C,BRKF
015E CD0500                CALL    BDOS
0161 C1D1E1                POP B! POP D! POP H; ENVIRONMENT RESTORED
```

```
0164 C9                         RET
                        ;
                        PCHAR: ;PRINT A CHARACTER
0165 E5D5C5                     PUSH H! PUSH D! PUSH B; SAVED
0168 0E02                       MVI     C,TYPEF
016A 5F                         MOV     E,A
016B CD0500                     CALL    BDOS
016E C1D1E1                     POP B! POP D! POP H; RESTORED
0171 C9                         RET
                        ;
                        CRLF:
0172 3E0D                       MVI     A,CR
0174 CD6501                     CALL    PCHAR
0177 3E0A                       MVI     A,LF
0179 CD6501                     CALL    PCHAR
017C C9                         RET
                        ;
                        ;
                        PNIB:   ;PRINT NIBBLE IN REG A
017D E60F                       ANI     0FH     ;LOW 4 BITS
017F FE0A                       CPI     10
0181 D28901                     JNC     P10
                        ;       LESS THAN OR EQUAL TO 9
0184 C630                       ADI     '0'
0186 C38B01                     JMP     PRN
                        ;
                        ;       GREATER OR EQUAL TO 10
0189 C637               P10:    ADI     'A' - 10
018B CD6501             PRN:    CALL    PCHAR
018E C9                         RET
                        ;
                        PHEX:   ;PRINT HEX CHAR IN REG A
018F F5                         PUSH    PSW
0190 0F                         RRC
0191 0F                         RRC
0192 0F                         RRC
0193 0F                         RRC
0194 CD7D01                     CALL    PNIB    ;PRINT NIBBLE
0197 F1                         POP     PSW
0198 CD7D01                     CALL    PNIB
019B C9                         RET
                        ;
                        ERR:    ;PRINT ERROR MESSAGE
                        ;       D,E ADDRESSES MESSAGE ENDING WITH "$"
019C 0E09                       MVI     C,PRINTF        ;PRINT BUFFER FUNCTION
019E CD0500                     CALL    BDOS
01A1 C9                         RET
                        ;
                        ;
                        GNB:    ;GET NEXT BYTE
01A2 3A1802                     LDA     IBP
01A5 FE80                       CPI     80H
01A7 C2B301                     JNZ     G0
                        ;       READ ANOTHER BUFFER
                        ;
```

```
                   ;
01AA CDCE01                CALL     DISKR
01AD B7                    ORA      A          ;ZERO VALUE IF READ OK
01AE CAB301                JZ       G0         ;FOR ANOTHER BYTE
                   ;       END OF DATA, RETURN WITH CARRY SET FOR EOF
01B1 37                    STC
01B2 C9                    RET
                   ;
                   G0:     ;READ THE BYTE AT BUFF+REG A
01B3 5F                    MOV      E,A        ;LS BYTE OF BUFFER INDEX
01B4 1600                  MVI      D,0        ;DOUBLE PRECISION INDEX TO DE
01B6 3C                    INR      A          ;INDEX=INDEX+1
01B7 321802                STA      IBP        ;BACK TO MEMORY
                   ;       POINTER IS INCREMENTED
                   ;       SAVE THE CURRENT FILE ADDRESS
01BA 218000                LXI      H,BUFF
01BD 19                    DAD      D
                   ;       ABSOLUTE CHARACTER ADDRESS IS IN HL
01BE 7E                    MOV      A,M
                   ;       BYTE IS IN THE ACCUMULATOR
01BF B7                    ORA      A          ;RESET CARRY BIT
01C0 C9                    RET
                   ;
                   SETUP: ;SET UP FILE
                   ;       OPEN THE FILE FOR INPUT
01C1 AF                    XRA      A          ;ZERO TO ACCUM
01C2 327C00                STA      FCBCR      ;CLEAR CURRENT RECORD
                   ;
01C5 115C00                LXI      D,FCB
01C8 0E0F                  MVI      C,OPENF
01CA CD0500                CALL     BDOS
                   ;       255 IN ACCUM IF OPEN ERROR
01CD C9                    RET
                   ;
                   DISKR: ;READ DISK FILE RECORD
01CE E5D5C5                PUSH H! PUSH D! PUSH B
01D1 115C00                LXI      D,FCB
01D4 0E14                  MVI      C,READF
01D6 CD0500                CALL     BDOS
01D9 C1D1E1                POP B! POP D! POP H
01DC C9                    RET
                   ;
                   ;       FIXED MESSAGE AREA
                   SIGNON:
01DD 46494C4520            DB       'FILE DUMP MP/M VERSION 1.0$'
                   OPNMSG:
01F8 0D0A4E4F20            DB       CR,LF,'NO INPUT FILE PRESENT ON DISK$'

                   ;       VARIABLE AREA
0218               IBP:    DS       2          ;INPUT BUFFER POINTER
021A               OLDSP: DS        2          ;ENTRY SP VALUE FROM CCP
                   ;
                   ;       STACK AREA
021C                       DS       64         ;RESERVE 32 LEVEL STACK
                   STKTOP:
                      ;
 Ø25C                         END
ØØØØ  BASE        ØØØ5 BDOS         Ø159 BREAK        ØØØB BRKF         ØØ8Ø BUFF
ØØØ1  CONS        ØØØD CR           Ø172 CRLF         Ø1CE DISKR        Ø19C ERR
ØØ5C  FCB         ØØ7C FCBCR        ØØ5C FCBDN        ØØ5D FCBFN        ØØ65 FCBFT
ØØ7D  FCBLN       ØØ6B FCBRC        ØØ68 FCBRL        Ø151 FINIS        Ø1B3 GØ
Ø123  GLOOP       Ø1A2 GNB          Ø218 IBP          ØØØA LF           Ø144 NONUM
Ø21A  OLDSP       ØØØF OPENF        Ø11B OPENOK       Ø1F8 OPNMSG       Ø189 P1Ø
Ø165  PCHAR       Ø18F PHEX         Ø17D PNIB         ØØØ9 PRINTF       Ø18B PRN
ØØ14  READF       Ø1C1 SETUP        Ø1DD SIGNON       Ø25C STKTOP       ØØØ2 TYPEF
```

Sample Resident System Process

This appendix contains a sample resident system process. It illustrates the required structure of a resident system process as well as the BDOS/XDOS access mechanism.

The first two bytes of a resident system process will contain the address of the BDOS/XDOS entry point. The address is filled in by the loader, providing a simple means for a resident system process to access the BDOS/XDOS by loading HL from the base of the program and then executing a PCHL instruction.

The process descriptor for the resident system process must immediately follow the address of the BDOS/XDOS entry point. Observe the manner in which the process descriptor is initialized in the example. The DS´s are used where storage is simply allocated. The DB´s and DW´s are used where data in the process descriptor must be initialized. Note that the stack pointer field of the process descriptor points to the address immediately following the stack allocation. The actual process entry point is contained at that address.

The procedure to produce a resident system process file closely follows that illustrated in the previous appendix on page relocatable programs. The only exception to the procedure is that the GENMOD output file should have a type of ´RSP´ rather than ´PRL´.

```
                            PAGE     0
                            TITLE    'Type File on Console'
                    ;       FILE TYPE PROGRAM, READS AN INPUT FILE AND PRINTS
                    ;         IT ON THE CONSOLE
                    ;
                    ;       COPYRIGHT (C) 1979
                    ;       DIGITAL RESEARCH
                    ;       P.O. BOX 579
                    ;       PACIFIC GROVE, CA  93950
                    ;
0000                        ORG      0000H              ; STANDARD RSP START

001A =              CTLZ    EQU      1AH                ; CONTROL-Z USED FOR EOF

0002 =              CONOUT  EQU      2                  ; BDOS CONOUT FUNCTION #
0009 =              PRINTF  EQU      9                  ;  ""    PRINT BUFFER
0014 =              READF   EQU      20                 ; READ NEXT RECORD
000F =              OPENF   EQU      15                 ; OPEN FCB
0098 =              PARSEFN          EQU      152               ; PARSE FILE NAME
0086 =              MKQUE   EQU      134                ; MAKE QUEUE
0089 =              RDQUE   EQU      137                ; READ QUEUE
0091 =              STPRIOR          EQU      145               ; SET PRIORITY
0093 =              DETACH  EQU      147                ; DETACH CONSOLE

                    ;
                    ; BDOS ENTRY POINT ADDRESS
                    BDOSADR:
0000                        DS       2                  ; IDR WILL FILL THIS IN
                    ;
                    ; TYPE PROCESS DESCRIPTOR
                    ;
                    TYPEPD:
0002 0000                   DW       0                  ; LINK
0004 00                     DB       0                  ; STATUS
0005 0A                     DB       10                 ; PRIORITY (INITIAL)
0006 0201                   DW       STACK+46           ; STACK POINTER
0008 5459504520             DB       'TYPE    '         ; NAME
                    PDCONSOLE:
0010                        DS       1                  ; CONSOLE
0011                        DS       1                  ; MEMSEG
0012                        DS       2                  ; B
0014                        DS       2                  ; THREAD
0016 2501                   DW       BUFF               ; DISK SET DMA ADDRESS
0018                        DS       1                  ; USER CODE & DISK SELECT
0019                        DS       2                  ; DCNT
001B                        DS       1                  ; SEARCHL
001C                        DS       2                  ; SEARCHA
001E                        DS       2                  ; SCRATCH

                    ;
                    ; TYPE LINKED QUEUE CONTROL BLOCK
                    ;
                    TYPELQCB:
0020 0000                   DW       0                  ; LINK
0022 5459504520             DB       'TYPE    '         ; NAME
```

```
002A 4800              DW      72                ; MSGLEN
002C 0100              DW      1                 ; NMBMSGS
002E                   DS      2                 ; DQPH
0030                   DS      2                 ; NQPH
0032                   DS      2                 ; MH
0034                   DS      2                 ; MT
0036                   DS      2                 ; BH
0038                   DS      74                ; BUF (72 + 2 BYTE LINK)

                ;
                ; TYPE USER QUEUE CONTROL BLOCK
                ;
                TYPEUSERQCB:
0082 2000              DW      TYPELQCB          ; POINTER
0084 8600              DW      FIELD             ; MSGADR

                ;
                ; FIELD FOR MESSAGE READ FROM TYPE LINKED QCB
                ;
                FIELD:
0086                   DS      1                 ; DISK SELECT
                CONSOLE:
0087                   DS      1                 ; CONSOLE
                FILENAME:
0088                   DS      72                ; MESSAGE BODY

                ;
                ; PARSE FILE NAME CONTROL BLOCK
                ;
                PCB:
00D0 8800              DW      FILENAME          ; FILE NAME ADDRESS
00D2 0401              DW      FCB               ; FILE CONTROL BLOCK ADDRESS

                ;
                ; TYPE STACK & OTHER LOCAL DATA STRUCTURES
                ;
                STACK:
00D4                   DS      46                ; 23 LEVEL STACK
0102 A901              DW      TYPE              ; PROCESS ENTRY POINT

0104            FCB:   DS      33                ; FILE CONTROL BLOCK

0125            BUFF:  DS      128               ; FILE BUFFER

                ;
                ; BDOS CALL PROCEDURE
                ;
                BDOS:
01A5 2A0000            LHLD    BDOSADR           ; HL = BDOS ADDRESS
01A8 E9                PCHL

                ;
                ; TYPE MAIN PROGRAM
                ;
                TYPE:
```

```
01A9 0E86               MVI     C,MKQUE
01AB 112000             LXI     D,TYPELQCB
01AE CDA501             CALL    BDOS            ; MAKE TYPELQCB
01B1 0E91               MVI     C,STPRIOR
01B3 11C800             LXI     D,200
01B6 CDA501             CALL    BDOS            ; SET PRIORITY TO 200

               FOREVER:
01B9 0E89               MVI     C,RDQUE
01BB 118200             LXI     D,TYPEUSERQCB
01BE CDA501             CALL    BDOS            ; READ FROM TYPE QUEUE
01C1 0E98               MVI     C,PARSEFN
01C3 11D000             LXI     D,PCB
01C6 CDA501             CALL    BDOS            ; PARSE THE FILE NAME
01C9 23                 INX     H
01CA 7C                 MOV     A,H
01CB B5                 ORA     L               ; TEST FOR 0FFFFH
01CC CA0E02             JZ      ERROR
01CF 3A8700             LDA     CONSOLE
01D2 321000             STA     PDCONSOLE       ; TYPEPD.CONSOLE = CONSOLE

01D5 0E0F               MVI     C,OPENF
01D7 110401             LXI     D,FCB
01DA CDA501             CALL    BDOS            ; OPEN FILE
01DD 3C                 INR     A               ; TEST RETURN CODE
01DE CA0E02             JZ      ERROR           ; IF IT WAS 0FFH, NO FILE
01E1 AF                 XRA     A               ; ELSE,
01E2 322401             STA     FCB+32          ;   SET NEXT RECORD TO ZERO
               NEW$SECTOR:
01E5 0E14               MVI     C,READF
01E7 110401             LXI     D,FCB
01EA CDA501             CALL    BDOS            ; READ NEXT RECORD
01ED B7                 ORA     A
01EE C21602             JNZ     DONE            ; EXIT IF EOF OR ERROR

01F1 212501             LXI     H,BUFF          ; POINT TO DATA SECTOR
01F4 0E80               MVI     C,128           ; GET BYTE COUNT
               NEXT$BYTE:
01F6 7E                 MOV     A,M             ; GET THE BYTE
01F7 5F                 MOV     E,A             ; SAVE IN E
01F8 FE1A               CPI     CTLZ
01FA CA1602             JZ      DONE            ; EXIT IF EOF
01FD C5                 PUSH    B               ; SAVE BYTE COUNTER
01FE E5                 PUSH    H               ; SAVE ADDRESS REGISTER
01FF 0E02               MVI     C,CONOUT
0201 CDA501             CALL    BDOS            ; WRITE CONSOLE
0204 E1                 POP     H               ; RESTORE POINTER
0205 C1                 POP     B               ; AND COUNTER
0206 23                 INX     H               ; BUMP POINTER
0207 0D                 DCR     C               ; DCR BYTE COUNTER
0208 C2F601             JNZ     NEXT$BYTE       ; MORE IN THIS SECTOR
020B C3E501             JMP     NEW$SECTOR      ; ELSE, WE NEED A NEW ONE

               ERROR:
020E 111E02             LXI     D,ERR$MSG       ; POINT TO ERROR MESSAGE
```

```
 0211 0E09              MVI     C,PRINTF        ; GET FUNCTION CODE TO PRINT
 0213 CDA501            CALL    BDOS

               DONE:
 0216 0E93              MVI     C,DETACH
 0218 CDA501            CALL    BDOS            ; DETACH THE CONSOLE
 021B C3B901            JMP     FOREVER

               ERR$MSG:
 021E 0D0A46696C        DB      0DH,0AH,'File Not Found or Bad File Name$'

 0240                   END
0000 BDOSADR   01A5 BDOS      0125 BUFF      0002 CONOUT    0087 CONSOLE
001A CTLZ      0093 DETACH    0216 DONE      021E ERRMSG    020E ERROR
0104 FCB       0086 FIELD     0088 FILENAME  01B9 FOREVER   0086 MKQUE
01E5 NEWSECTOR 01F6 NEXTBYTE  000F OPENF     0098 PARSEFN   00D0 PCB
0010 PDCONSOLE 0009 PRINTF    0089 RDQUE     0014 READF     00D4 STACK
0091 STPRIOR   0020 TYPELQCB  01A9 TYPE      0002 TYPEPD
0082 TYPEUSERQCB
```