```
**********************************************
*                                            *
*     MP/M 1.1  -  Application Note #01      *
*                                            *
**********************************************
```

Suppressing the MP/M Loader Display

(Note: This application note is a revision of MP/M 1.0 Application Note #03.)

When the MP/M 1.1 loader reads and relocates the MPM.SYS file, a load map is displayed on console #0. In some applications it may be desirable to suppress this display.

The MP/M loader display can be suppressed as follows:

1.) Edit the LDRBIOS.ASM file, replacing the console output code with a RET instruction.

```
; Loader BIOS jump vector:

        ...
        jmp     conout
        ...


conout:
        ret
```

2.) Follow the steps provided in the MP/M 1.0 Application Note #01 or the MP/M User's Guide (2nd printing) to integrate the new LDRBIOS.HEX into the MPMLDR.COM file.

3.) Follow the steps provided in the MP/M 1.0 Application Note #02 or the MP/M User's Guide (2nd printing) to update the cold start loader with the new MPMLDR.COM file.

```
**********************************************
*                                            *
*     MP/M 1.1  -  Application Note #02      *
*                                            *
**********************************************
```

Customizing the MP/M 1.1 Sign-On Message

(Note: This application note is a revision of MP/M 1.2 Application Note #04.)

The MP/M 1.1 sign-on message can be customized (within the constraints identified in the Digital Research Software License Agreement) to include an ASCII string of your choice which is up to 3E characters. The string must be terminated with an ASCII '$' character.

The following steps can be used to customize the MP/M 1.1 sign-on message while running under MP/M 1.1:

```
A>DDT XDOS.SPR
        (NEXT should be 2500E)

  -S230
        0230 0D  <- at this point up to 3E characters can be
                    inserted in hexadecimal, terminated by a
                    '$' character (24H).

  -IXDOS.SPR
  -W48
  -G0
```

```
**********************************************
*                                            *
*     MP/M 1.1  -  Application Note #03      *
*                                            *
**********************************************
```

CLI Default to System Drive 'x' for PRL & COM Files

The purpose of this XDOS patch is to default to drive 'x' (where 'x' is drive A: to P:) if a file to be executed cannot be found on the currently logged drive. Drive 'x' is searched for files of type PRL and COM.

The following steps can be used to install this patch while running under MP/M 1.1:

```
0A>DDT XDOS.SPR
        (NEXT should be 2500H)

-A15AA
       15AA CALL [Patch Area]

            (Note: The MP/M 1.1 XDOS patch area begins
                   at 16F0H relative to the XDOS base.
                   Make certain that you will not be
                   overlaying some other patch. The
                   patch area extends to 175FH.)

-A [Patch Area]

            PUSH B        ; save FCB address
            CALL XXXX     ; copy of the call @ 15AAH
            POP  H        ; HL = FCB address
            CPI  FF
            RNZ           ; ret if open *.COM OK
            MVI  A,sdrv   ; where sdrv is default sys drv
            CMP  M
            MVI  A,FF
            RZ            ; return error if tried already
            MVI  M,sdrv   ; where sdrv is default sys drv
            POP  D        ; discard the return addr
            LXI  D,9
            DAD  D        ; HL -> file type field of FCB
            MVI  M,20     ; type must start with a blank
            JMP  1377     ; jump to try PRL & COM again

-B  [Perform the required bit set/reset for above patch]

-IXDOS.SPR
-W48
-G0
```

```
**************************************************
*                                                *
*      MP/M 1.1  -  Application Note #04         *
*                                                *
**************************************************
```

Setting the Raw Console Mode

In some application programs it is desirable accept raw input from the console. Raw input implies that the operating system takes no action on any special characters, such as ^C.

An application program may place itself into a raw console input mode by executing the following code sequence:

```
...

MVI  C,9CH
CALL XDOS     ; get process descriptor address
LXI  D,8
DAD  D
MOV  A,M
ORI  80H      ; turn 'on' the high order bit of first
MOV  M,A      ;  character in the process name
...
```

```
*********************************************
*                                           *
*     MP/M 1.1  -  Application Note #05     *
*                                           *
*********************************************
```

Obtaining the Printer Mutual Exclusion Queue Message

Application programs which require use of the printer during their execution should properly obtain the printer mutual exclusion message prior to sending characters to the printer. This application note shows the code sequences required to obtain and release the printer mutual exclusion queue message.

```
...

LXI  D,UQCB
MVI  C,OPENQUE
CALL XDOS       ; open the 'MXList' queue
INR  A
JZ   ERRNOSUCHQUE
...
```

Either of the two following code sequences can be executed after the queue has been opened:

```
...

;  Code to hang and wait until the printer is free
;
LXI  D,UQCB
MVI  C,READQUE
CALL XDOS       ; perform unconditional read queue
...

      - or -
...

;  Code to test and display message if printer is busy
;
LXI  D,UQCB
MVI  C,CREADQUE
CALL XDOS       ; perform conditional read queue
INR  A
JZ   PRINTERBUSY  ; display printer busy message
...
```

Then the printer can be used. If the application program terminates, it does not need to return the mutual exclusion message because that is done automatically at termination.

The following code sequence can be used to write the mutual exclusion message back to the queue when the process is done with the printer:

```
...

LXI  D,UCCB
MVI  C,WRITECUE
CALL XDOS       ; perform the unconditional write queue
...
```

The following data structure is required by the code sequences shown above:

```
;  Data Segment
;
UCCB:
     DS   2          ; pointer, filled in by open queue
     DW   0          ; buffer pointer, not used
     DB   'MXList '  ; queue name, padded to 8 chars
```

```
**********************************************
*                                            *
*     MP/M 1.1  -  Application Note #06      *
*                                            *
**********************************************
```

Changing PRL File Minimum Buffer Size Requirements

The minimum buffer size requirements for PRL files can be changed by following the procedure outlined in this application note. It may be desirable to require a larger default buffer for a program such as the editor. The following steps show how to change the minimum buffer size requirements for ED from 4k to 8k bytes:

```
0A>DDT ED.PRL
[MP/M] DDT VERS 1.1
NEXT  PC
1F00 0100
-S104
0104 00
0105 12 20
0106 .
-V1F00
003C
-IED.PRL
-W3C
-G0
```

Note: bytes 4 and 5 of the PRL header record (relative to the base) contain the low and high order bytes of the minimum buffer size specification.

```
*********************************************
*                                           *
*     MP/M 1.1  -  Application Note #07     *
*                                           *
*********************************************
```

Accessing the Internal MP/M 1.1 TOD

In some applications it is desirable to access the internal MP/M 1.1 time and date fields for the purposes of setting initial values. The following code sequence could be executed at the end of your MP/M XIOS system initialization procedure. It should be placed at the end because the XDOS call to obtain the system data page address has a side effect of enabling interrupts.

```
...

MVI  C,98H
CALL XDOS       ; obtain the system data page address
                ; *** warning ***
                ; the XDOS call enables interrupts

LXI  D,22FCH
DAD  D          ; hl -> pointer -> TOD
MOV  E,M
INX  H
MOV  D,M        ; de -> TOD

...
```

```
*****************************************
*                                       *
*     MP/M 1.1  -  Application Note #28 *
*                                       *
*****************************************
```

DMA Disk Controllers with Banked Memory Systems

Special care must be taken in bank switched memory systems that have DMA disk controllers. The reason is that bank switching must not be allowed during the transfer of data from the disk controller into a target bank. The following solutions can be used to avoid potential problems:

1.) The DMA from the disk controller can be only made into common memory and then copied from common memory into the desired actual user buffer. This is a reasonable technique in systems where deblocking is required. Sectors which are larger than 128 bytes are placed in a buffer in common memory and then the specified sector is transferred to the target buffer.

2.) The following measures must be taken if the DMA is to be made directly into the user buffer (not in common memory):

Prior to each DMA operation a DMA active flag must be set true and then reset following the DMA operation:

```
        ...

        MVI   A,FFH
        STA   DMACTVE

        ;  initiate DMA operation

        ;  perform flag wait or poll for operation complete

        XRA   A
        STA   DMACTVE
        ...
```

Another code sequence must be placed in the XIOS select memory procedure to ensure that the bank will not be switched during a DMA operation:

```
SELMEMORY:
        ...

        LDA   DMACTVE
        ORA   A
        JZ    OKTOSWITCH      ; jump if not in DMA operation
```

```
        ; Next, the bank to be switched can be compared with the
        ;  current bank.  If it matches, the DMA operation will
        ;  not be affected.

        JZ   OKTOSWITCH     ; no bank change required

        ; A  new  bank is  specified and a DMA operation is  in
        ;  progress.  A busy wait must now be performed to wait
        ;  until the DMA operation is complete.

        ; *** warning ***

        ; The selmemory call is made from inside the dispatcher
        ;   therefore interrupts are disabled and nothing  must
        ;   be done which could force a dispatch.

    BUSYWAIT:
        IN   DMASTATUSPORT  ; This is a "BUSY-WAIT" !
        ANI  DMADONE
        JZ   BUSYWAIT       ; loop until the DMA is complete
```

Then drop into the remaining select memory procedure:

```
    OKTOSWITCH:
        ...

        ...

        RET
```

```
**********************************************
*                                            *
*     MP/M 1.1  -  Application Note #09      *
*                                            *
**********************************************
```

BDOS Disk Error Pause

While it is possible to place a patch directly into the BDOS itself to pause on disk errors, a more desirable solution would be to detect the error condition in the user written XIOS. At that point the error condition could be returned, forcing the termination of the process which made the BDOS call, or no error condition returned in case the user wishes to accept the error condition and continue.

A note of caution must be added at this point. When XIOS disk driver code is being executed, the calling process owns the disk mutual exclusion queue message. Thus, all other processes are denied access to the BDOS disk file functions until the calling process either terminates or returns from the XIOS. What this means is that the everybody else STOPS until the required response is made at the keyboard.

It also must be noted that BDOS calls must not be made during the error handling process.

The following code sequence shows how to obtain the console number of the calling process for the purposes of displaying an error message and obtaining a response.

```
;  Disk error encountered within the XIOS
;
     MVI  C,99H
     CALL XDOS            ; get console number
     MOV  D,A
     PUSH D               ; save console number on stack
;
;  Display error message and ask for response
;
     LXI  H,MSGADR
LOOP:
     MOV  C,M
     POP  D
     PUSH D
     PUSH H
     CALL CONOUT
     POP  H
     INX  H
     MOV  A,M
     ORA  A
     JNZ  LOOP
```

```
;
; Obtain response
;
     PCP  D
     CALL CONIN
     CPI  'C'
     JZ    ACCEPTERROR          ;  accept the error,  do  not
                                ;   return an error code from
                                ;   from the XIOS

     JMP  RTNERROR              ;   jump to return the normal
                                ;    error  code, which  will
                                ;    terminate the process

;
;  Data Area
;
MSGADR:
     DB   'ENTER "C" TO ACCEPT ERROR AND CONTINUE',0
```

```
**********************************************
*                                            *
*     MP/M 1.1  -  Application Note #10      *
*                                            *
**********************************************
```

Forcing Printer Mutual Exclusion Within the XIOS

In some applications it may be desirable to force ownership of the printer mutual exclusion message prior to allowing any list output to be performed. This can be implemented within the user written XIOS as is shown below.

The code segment which follows should be placed at the XIOS list entry point. It performs the following functions:

1.) Initialization code is executed when the first character is sent to the printer following a cold start. The MP/M ready list root address is obtained and the 'MXList' queue is opened.

2.) Then, each time a character is sent to the printer a test is performed to determine if the printer mutual exclusion message has been consumed. If no other process is using the printer, the mutual exclusion message is obtained for the calling process and the character is printed. If the mutual exclusion message has been consumed, the following steps are performed.

3.) If the printer mutual exclusion message is owned by a process with the same console (i.e. owned by the calling process), the character is printed. If the mutual exclusion message is owned by some process with a different console, the following steps are performed.

4.) A delay of 1/10 of a second is done and a check is made to determine if the printer is free. The delay is done to prevent a 'busy-wait' which would remain compute bound until the printer was free. If the printer mutual exclusion message is obtained by the conditional read queue, the character is printed. Otherwise, the following steps are performed.

5.) The console is checked for a ^C and if it has been entered the process is terminated. Otherwise, the process continues to spin in the loop at step 2 above.

6.) If a ^Z character is sent to the printer, the calling process relinquishes the printer mutual exclusion message.

```
LIST:                                   ;XIOS LIST OUTPUT

;
; PATCH TO FORCE MXLIST QUEUE MESSAGE OWNERSHIP
;
        PUSH    B                       ;
        LDA     LISTINIT
        ORA     A
        JNZ     RELIST                  ; JUMP IF INIT DONE
        MVI     C,9AH
        CALL    XDOS                    ; GET SYSTEM DATA PG ADDR
        LXI     D,2ØFCH
        DAD     D
        MOV     E,M
        INX     H
        MOV     D,M                     ; DE -> MP/M INTERNAL DATA SEG
        LXI     H,2ØØ5H
        DAD     D                       ; HL -> RLR ADDR
        SHLD    RLRADR
        MVI     C,87H
        LXI     D,MXLUQCB
        CALL    XDOS                    ; OPEN 'MXLIST' QUEUE
        INR     A
        JZ      SYINITERR               ; JUMP IF OPEN QUEUE ERROR
        LHLD    MXLUQCB
        LXI     D,22
        DAD     D                       ; HL -> MXLQCB.CNT
        SHLD    MXLQCBCNT
        MVI     A,2FFH
        STA     LISTINIT                ; INDICATE THAT INIT DONE

RELIST:
        LHLD    RLRADR
        MOV     E,M
        INX     H
        MOV     D,M                     ; DE = PD ADDR
        LXI     H,2ØØEH
        DAD     D
        MOV     D,M                     ; D = CONSOLE #
        LHLD    MXLQCBCNT?
        DI                              ; EI WILL BE AT NEXT XDOS CALL
        MOV     A,M                     ; THIS MUST BE AN INDIVISIBLE
        ORA     A                       ;  TEST & SET SEQUENCE
        JZ      MSGTAKEN
        MVI     C,89H                   ; MXLIST QUEUE MSG IS FREE
        LXI     D,MXLUQCB
        CALL    XDOS                    ; READ MXLIST QUEUE MESSAGE
        JMP     CONTLIST
```

```
MSGTAKEN:
        INX     H
        INX     H
        MOV     C,M
        INX     H
        MOV     B,M             ; BC -> PD ADDR OF MSG OWNER
        LXI     H,0007H
        DAD     B               ; HL -> PD.CONSOLE
        MOV     A,M
        CMP     D
        JZ      CONTLIST        ; JUMP IF PRINTER IS OURS
        POP     B
        MOV     B,D
        PUSH    B               ; SAVE CONSOLE #
        MVI     C,9DH
        LXI     D,3
        CALL    XDOS            ; DELAY 1/10 SECOND
        MVI     C,3AH
        LXI     D,MXLUQCB
        CALL    XDOS            ; COND. READ MXLIST QUE MSG
        ORA     A
        JZ      CONTLIST        ; JUMP IF MESSAGE OBTAINED
        POP     B
        MOV     D,B
        PUSH    B               ; D = CONSOLE #
        CALL    CONST           ; CONSOLE STATUS CHECK
        ORA     A
        JZ      RELIST
        POP     B
        MOV     D,B
        PUSH    B               ; D = CONSOLE #
        CALL    CONIN           ; GET CHARACTER, RAW INPUT
        CPI     3               ;  ^C ?
        JNZ     RELIST          ; CONTINUE LOOP IF NO ABORT
        MVI     C,0
        CALL    XDOS            ; TERMINATE THE PROCESS ON ^C
        JMP     RELIST          ; TERMINATE MAY FAIL IF SYS PROC

CONTLIST:
        POP     B
        MVI     A,1AH
        CMP     C               ; ^Z ?
        JNZ     NOTCTLZ
        MVI     C,03BH
        LXI     D,MXLUQCB
        JMP     XDOS            ; WRITE MXLIST QUEUE MESSAGE
;       RET

NOTCTLZ:
        ;  continue with normal printer output, character in C
```

```
;
; PATCH DATA STRUCTURES
;
LISTINIT:
        DB      0                       ; INIT DONE FLAG
RLRADR: DS      2                       ; MP/M READY LIST ROOT PTR
MXLQCBCNT:
        DS      2                       ; PTR TO MXLQCB.CNT FIELD
MXLUQCB:                                ; USER QUEUE CONTROL BLOCK
        DS      2                       ; QCB POINTER
        DW      0                       ; BUFFER POINTER
        DB      'MXList '               ; QUEUE NAME
```

```
*********************************************
*                                           *
*     MP/M 1.1  -  Application Note #11     *
*                                           *
*********************************************
```

Improving Disk Read/Write Performance with BNKBDOS

This application note should allow you to improve your disk read/write performance when using the BNKBDOS by 15 to 35%. The actual amount of improvement will vary, depending on how well your skew (interleave) tables are optimized for your disk drives. In fact, after installing this patch you are strongly encouraged to perform careful tests to determine the optimal skew for your drives.

The effect of this patch is to enable the ODOS to make direct SELMEMORY calls to perform the bankswitching required to access the BNKBDOS, rather than to simply change the memory segment index of the calling process and perform a dispatch to force the bankswitching. The reduction in overhead is at least 4 milliseconds (4 Mhz Z-80) per disk sector read/written.

Changes are required to both the ODOS.SPR and XDOS.SPR files. Note that XDOS patch area used must be examined carefully to ensure that you have not installed other custom patches which conflict with this one.

ODOS.SPR Patches:

```
0A>ren odos.old=odos.spr
0A>ddt odos.old
[MP/M] DDT VERS 1.1
NEXT  PC
0B00 0100
-s497
0497 FA 20
0498 FF f8
0499 C1 .
-s4ab
04AB FA 20
04AC FF f8
04AD D1 .
-s93f
093F FA 20
0940 FF f8
0941 21 .
-s96f
096f FA 20
0970 FF f8
0971 C1 .
```

```
-s980
0980 FA 20
0981 FF f8
0982 F1 .
-iodos.spr
-w14
-g0
```

XDOS.SPR Patches:

```
0A>ren xdos.old=xdos.spr
0A>ddt xdos.old
[MP/M] DDT VERS 1.1
NEXT  PC
2500 0100
-d1920,1935
1920 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
1930 00 00 00 00 00 00
-a1920
1920  mov a,m
1921  inr a
1922  rz
1923  dcr a
1924  add a
1925  add a
1926  lxi b,18fd
1929  add c
192A  mov c,a
192B  mvi a,0
192D  mov e,a
192E  adc b
192F  mov b,a
1930  di
1931  call 2733
1934  ei
1935  ret
1936  .
-b1828,1
-b1833,1
-ixdos.spr
-w48
-g0
```

After completing the patches shown above, you must perform a GENSYS to include the new XDOS.SPR and ODOS.SPR files in your MPM.SYS file.

```
*********************************************
*                                           *
*     MP/M 1.1  -  Application Note #12     *
*                                           *
*********************************************
```

Accessing MP/M's Internal Data Segment

This application note is intended to provide you with information regarding the location of critical variables contained in MP/M's internal data segment. The information may be useful in some application programs, however, it must be accessed with caution. The information might also be useful in debugging a system by permitting you to access the ready list through the Ready List Root (RLR), both at run time as well as in a post-mortem dump.

The following example illustrates the technique to access the Ready List Root:

```
;       MP/M Internal Data Segment Offsets
;
ostod           equ     0000h   ; time of day
osrlr           equ     0005h   ; ready list root
osdlr           equ     0007h   ; delay list root
osdrl           equ     0009h   ; dispatcher ready list
osplr           equ     000Bh   ; poll list root
osslr           equ     000Dh   ; swap list root (not used)
osqlr           equ     000Fh   ; queue list root
osthrdrt        equ     0011h   ; thread root
osnmbcns        equ     0013h   ; number of consoles
oscnsatt        equ     0014h   ; console attach table
oscnsque        equ     0034h   ; console queue
osnmbflags      equ     0054h   ; number of flags
ossysfla        equ     0055h   ; system flags
osnmbsegs       equ     0095h   ; number of memory segments
osmsegtbl       equ     0096h   ; memory segment table
ospdtbl         equ     00B6h   ; process descriptor table

sysdatadr       equ     154     ; get system data page addr
        ...

        mvi     c,sysdatadr
        call    xdos            ; HL = system data page
        lxi     d,00fch
        dad     d
        mov     e,m
        inx     h
        mov     d,m             ; DE = base of MP/M intrl dseg

        lxi     h,osrlr         ; HL = offset to Ready List Root
        dad     d
        ...                     ; HL = Addr of Ready List Root
```

```
*********************************************
*                                           *
*     MP/M 1.1  -  Application Note #13     *
*                                           *
*********************************************
```

Using the Send CLI Command XDOS Function

This application note is intended to show you how to properly make use of the Send CLI Command XDOS Function. This is a very powerful function as it can be used to implement a menu driven applications program. In the example that follows you should observe the following steps:

1) The priority of the calling program is changed so that it is higher (actually a lower value itself) than the TMP.

2) The console number of the calling program is obtained.

3) The console is assigned to the Command Line Interpreter.

4) The send CLI command function call is issued.

5) An attach console function call is made to get the console back after the program initiated has terminated.

6) Then the original priority of the calling program is restored to its original value (usually 200).

The following example is shown as segments of a menu driven program named MENU:

```
;
; XDOS Function Equate Table
;
setpriority     equ     145
attachconsole   equ     146
assignconsole   equ     149
sendCLIcommand  equ     150
getconsole      equ     153
```

```
MENU:
        ...

        mvi     e,190
        mvi     c,setpriority
        call    BDOS            ; set priority to 190
        mvi     c,getconsole
        call    BDOS            ; get console # in A reg
        sta     AssignPB        ; fill in
        sta     CLIcommand+1    ;         console fields
        lxi     d,AssignPB
        mvi     c,assignconsole
        call    BDOS            ; assign console to CLI
        inr     a
        jz      cannotassign    ; assign failed
        lxi     d,CLIcommand
        mvi     c,sendCLIcommand
        call    BDOS            ; send CLI command
        mvi     c,attachconsole
        call    BDOS            ; attach console
        mvi     e,200
        mvi     c,setpriority
        call    BDOS            ; set priority back to 200
        ...

AssignPB:
        db      $-$             ; console number
        db      'cli    '       ; name (cli is lower case)
        db      0
        ...

CLIcommand:
        db      0               ; default disk / user code
        db      $-$             ; console number
        db       this is an ASCII string terminated with a
                 null that is exactly as you would run the
                 program from the console. e.g.
                 'PIP LST:=MYPROG.LST[PT8]',0
        ...
```

```
*********************************************
*                                           *
*     MP/M 1.1  -  Application Note #14     *
*                                           *
*********************************************
```

Submit File Accessing from other than Drive A:

In its standard release form, the SUBMIT program of MP/M creates a temporary file on drive A: of the form X.SUB, where 'X' is the console at which the SUBMIT command was issued. Then the TMP for the console reads the X.SUB file from drive A: and executes the command lines.

The purpose of this application note is to show you how to patch both the SUBMIT.PRL and XDOS.SPR files to enable you to place the X.SUB files on other than drive A:. This would be desirable in situations where drive A: is a floppy disk, while drive E: is a hard disk.

SUBMIT.PRL Patch

```
0A>ddt submit.prl
[MP/M] DDT VERS 1.1
NEXT  PC
1180 0100
-s6f0
06F0 01 05     ; -or- any other drive, 05 = drive E:
06F1 24 .
-isubmit.prl
-w21
-g0
```

XDOS.SPR Patch

```
0A>ddt xdos.spr
[MP/M] DDT VERS 1.1
NEXT  PC
2500 0100
-sfae
0FAE 01 05    ; -or- any other drive, same as above
0FAF 24 .
-ixdos.spr
-w48
-g0
```

```
nofile:
        lxi     d,FCB
        mvi     c,makefile
        call    BDOS            ; make the $X$.SUB file
        ...

;
; Now, write the records into the the $X$.SUB file as
;  follows:
;               -one line of the submit file per record
;
;               -last record first (i.e. in reverse order
;                 that they are to be executed
;
;               -each record in the following form:
;
;                    [CNT][ASCII command line][NULL]
;                  where:  CNT = # chrs in cmnd ln, 1 byte
;                          ASCII command line <= 125 chrs
;                          NULL = zero, 1 byte
;
        ...
        ...

        lxi     d,FCB
        mvi     c,closefile
        call    BDOS            ; close the $X$.SUB file
        mvi     c,getsysdatadr
        call    BDOS            ; get system data page adr
        lda     console         ; retrieve the saved con #
        adi     80h             ; add offset to base of
        mov     e,a             ;  submit flag array
        mvi     d,0
        dad     d               ; DE = .submitflag(console);
        mvi     m,0ffh          ; set 'on' submit flag
        ...

; terminate the program

        ...

FCB:
        DB      1               ; disk drive, usually A:
        DB      '$X$    '       ; file name
        DB      'SUB'           ; file type
        DB      0               ; file extent
        DS      20              ;  remainder of FCB

console:
        ds      1               ; temp loc for console #

        ...
```

```
*********************************************
*                                           *
*     MP/M 1.1  -  Application Note #15     *
*                                           *
*********************************************
```

Creating a Submit File from an Applications Program

This application note is intended to show you how to create a submit file from an applications program and then how to force its execution and, if required, its termination. The example shown below illustrates the following steps:

1) Obtain the console number at which the program is executing.

2) Create the X.SUB file, where 'X' is the console number.

3) Set 'on' the appropriate submit flag in the submit flag array contained in the system data page.

```
;
; BDOS / XDOS Function Equate Table
;
closefile       equ     16
searchfirst     equ     17
deletefile      equ     19
makefile        equ     22

getconsole      equ     153
getsysdatadr    equ     154


        ...
        ...

        mvi     c,getconsole
        call    BDOS
        sta     console
        adi     '0'
        sta     FCB+2           ; put console # in fname
        lxi     d,FCB
        mvi     c,searchfirst
        call    BDOS            ; see if file there
        inr     a
        jz      nofile
        lxi     d,FCB
        mvi     c,deletefile
        call    BDOS            ; delete old version first
```

```
*********************************************
*                                           *
*     MP/M 1.1  -  Application Note #16     *
*                                           *
*********************************************
```

Using the BDOS of the MPMLDR and Loading Additional Files Using the MPMLDR

This application note is intended to provide you with additional information about the MPMLDR such that you can make use of its LDRBDOS in your MP/M SYSINIT entry point code.

Using the BDOS of the MPMLDR

The BDOS of the MPMLDR is called the LDRBDOS and its entry point is located at 0D06H. The calling conventions for the LDRBDOS are identical to that of the BDOS. That is, the function code is passed in the C register while parameters are passed in the DE register pair, etc. Remember that the LDRBDOS and the user customized LDRBIOS do not necessarily have the code for such fuctions as console input and disk writes.

Loading Additional Files Using the MPMLDR

In certain environments it may be desirable to load and relocate additional files underneath the BNKBDOS.SPR. This portion of the application note describes an entry point of the MPMLDR, called LOADSPR, that may be used for such purposes. The LOADSPR procedure would be called from MP/M system initialization in the user supplied SYSINIT code.

The following example outlines the steps involved to use the LOADSPR procedure:

```
;
LOADSPR equ     05c4h   ; LOADSPR procedure address
LDRBDOS equ     0d06h   ; LDRBDOS entry point address
Curtop  equ     0b6ch   ; current top address

SYSINIT:
        ...
        ...

        ; At this point Curtop contains the address of the
        ; base of BNKBDOS, the SPR file to be loaded will
        ; be placed immediately below Curtop and Curtop will
        ; then be updated.  Curtop can be set before calling
        ; LOADSPR.
```

```
...

lxi     d,SPRFCB
mvi     c,openfile
call    LDRBDOS         ; open the SPR file
inr     a
jz      filenotfound
lxi     b,0
lxi     d,SPRFCB
call    LOADSPR         ; load the SPR file below
...                     ;  Curtop
                        ; Curtop is reset to base of
                        ;  the loaded SPR
```

Terminating a Submit File Job

At times it may be necessary to terminate the operation of a submit job. This can be done only by zeroing a submit flag which is maintained in the CONSOLE.DAT region of memory. The submit flag array in the system data page is used only to begin a submit job and is zeroed immediately after the TMP begins executing the submit job.

Locating and then zeroing the submit flag for console can be done as follows:

```
;
;  XDOS Function Equate Table
;
getconsole      equ     153
getsysdatadr    equ     154


        ...
        ...

        mvi     c,getconsole
        call    BDOS            ; get console #
        mov     b,a
        push    b               ; save console #
        mvi     c,getsysdatadr
        call    BDOS            ; get system data page adr
        inx     h
        mov     a,m             ; A = max consoles
        mvi     l,0eeh          ; L = offset to submit flag
        pop     b               ; B = console #
        sub     b               ; # pgs down to console area
pagedcr:
        dcr     h               ; dcr page address
        dcr     a               ; dcr count
        jnz     pagedr
        mvi     m,0             ; zero the submit flag
        ...
```