CP/M-86™

Operating System

Release 1.1

Release Notes

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

CP/M-86™ Operating System

Release 1.1

Compiled February 1982

Thank you for purchasing the CP/M-86™ operating system package. Software included in this package is proprietary to Digital Research and contains internal serialization to allow unauthorized copies to be traced to their source. The Digital Research Software License Agreement defines the terms and conditions covering the use of CP/M-86. Please take time to carefully read this agreement. The enclosed Software Registration Card must be filled out and mailed to Digital Research before use of this software is authorized. Upon receipt of the Registration Card, your name will be placed on our CP/M-86 mailing list, so you will receive newsletters and update notices. Under the terms of the agreement, you are allowed to make back-up copies for your own use, but you are not allowed to make copies of software provided in this package for any third parties, including friends, relatives, or business associates.

The documentation for CP/M-86 consists of the following manuals:

CP/M-86 Operating System User's Guide

CP/M-86 Operating System Programmer's Guide

CP/M-86 Operating System System Guide

CP/M-86 Operating System Command Summary

Two diskettes are also included. The first disk contains the CP/M-86 operating system and the utility programs. The second disk contains the source files for programs and data files used in system regeneration. The following programs are on the first disk.

ASM86.CMD	8086 assembler
ASM86.COM	8080 version of ASM-86™ assembler
COPYDISK.CMD	Utility to copy entire diskette
CPM.H86	Hex file for CP/M-86 CCP and BDOS
CPM.SYS	CP/M® system file, loaded at cold start
DDT86.CMD	CP/M-86 debugger
ED.CMD	CP/M-86 program and text editor
GENCMD.CMD	CMD file generation utility

GENCMD.COM	8080 version of GENCMD
GENDEF.CMD	Diskdef file generator
GENDEF.COM	8080 version of GENDEF
HELP.CMD	Help utility
HELP.HLP	Data file for help utility
LDBDOS.H86	Loader BDOS hex file
LDBIOS.H86	Loader BIOS hex file
LDCOPY.CMD	Loader copy utility
LDCPM.H86	Loader main program hex file
LMCMD.CMD	CMD file generation utility
LMCMD.COM	8080 version of LMCMD
LOADER.CMD	ISBC™ 86/12 intermediate loader (used only with the standard Intel® system)
PIP.CMD	Peripheral Interchange Program
STAT.CMD	File and disk status utility
SUBMIT.CMD	Batch processing utility
TOD.CMD	Display and set time of day utility

The files with a filetype of CMD operate under CP/M-86. The files with a filetype of COM are included for cross development under CP/M-80™.

The second disk contains the following files.

BIOS.A86	Source file for the standard BIOS
CBIOS.A86	Source for the skeletal BIOS
COPYDISK.A86	Source for COPYDISK.CMD
DEBLOCK.LIB	Blocking/deblocking algorithms
LDBIOS.A86	Source for LDBIOS.CMD
LDCOPY.A86	Source for LDCOPY.CMD
LDCPM.A86	Source for LDCPM.CMD
RANDOM.A86	Sample A86 program using BDOS calls
ROM.A86	Source file for the ISBC 86/12 boot ROM
SINGLES.DEF	Diskdef input to the GENDEF utility
SINGLES.LIB	Output from the GENDEF utility
TBIOS.A86	Source for track buffered BIOS
TRACK.A86	Skeletal source for track buffering
8087.LIB	Code macro library for 8087

Note: The DEBLOCK.LIB file is included for your reference. Any specific application might require modifications.

CP/M-86™ Operating System

Version 1.1

Enhancements

Digital Research is pleased to supply you with CP/M-86™ Update Version 1.1. This version of our single-user 8086/8088 operating system has many enhancements we feel you, as an end user, can appreciate:

1.1 Update Features:

- A HELP facility has been added.
- The user facility has been enhanced to allow you to access system files in user area 0.
- Program chaining lets one program chain to the next without operator intervention.
- All utilities are reduced on size and execution time.
- A DIRS command added to the CCP allows display of system files.

Utility Enhancements Overview:

ED reads and writes to, and deletes INCLUDE files. You can now specify an input and an output file. ED attempts to recover from DISK FULL errors by erasing the backup file and retrying. ED backspaces past the beginning of the line.

ED allows specification of different input and output file specifications. Specify the output file, if different from the input file, after the input file, as shown below:

ED <input filespec> <output drive or filespec>

If you specify an output file, no backup file is created. This allows the input file to be on a write-protected disk.

ASM-86™ symbols are now alphabetized in the SYM file. About 5.5K more of space is available for the symbol table.

PIP can be utilized between user areas. The SPARSE file copy is supported.

GENCMD now allows you to create a file without a header record, and create a file with a prefilled memory. Do this by including .NOHEAD in the command tail.

DDT-86[T.M.] compares memory facility.

SID-86[T.M.] is available to support CP/M-86.

STAT checks for the existence of duplicate block assignments (an invalid directory) and displays an error message if an allocation conflict is discovered. You should erase the file containing the conflict, and reset the disk with a CTRL-C. Use,

```
STAT *.*
```

to obtain the duplicate block check.

Please contact Digital Research Technical Support at (408) 375-6262 if you have technical difficulties. Send us your registration card, and you will automatically receive CP/M-86 application notes and patches directly from Digital Research.

CP/M-86™ Operating System

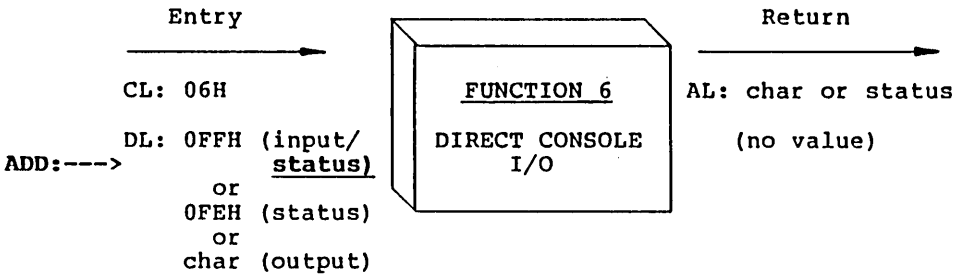
SYSTEM GUIDE

Correction to the First Printing - 1981

Compiled February 1, 1982

PAGE 27

To the FUNCTION 6 DIRECT CONSOLE I/O BLOCK,



The second paragraph following FUNCTION 6 should read:

Upon entry to Function 6, register DL contains either (1) a hexadecimal FF denoting a CONSOLE input/status request, or (2) a hexadecimal FE denoting a console status request, or (3) an ASCII character to be output to CONSOLE where CONSOLE is the logical console device. If the input value is FF, then Function 6 checks to see if a character is ready. If a character is ready, Function 6 returns the character in AL; otherwise Function 6 returns a zero in AL. If the input value is FE and no character is ready, then Function 6 returns AL = 00; otherwise, AL = FF. If the input value in DL is not FE or FF, then Function 6 assumes that DL contains a valid ASCII character which is sent to the console.

You cannot use Function 6 with FF or FE in combination with either Function 1 or Function 11. Function 1 is used in conjunction with Function 11. Function 6 must be used independently.

CP/M-86™ Operating System

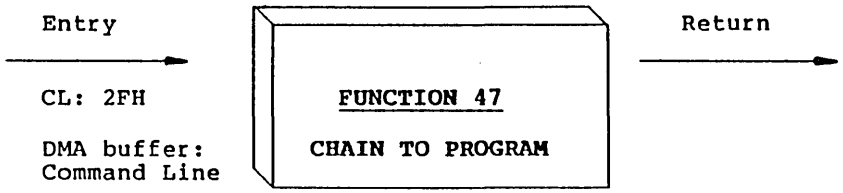
SYSTEM GUIDE

Enhancements to the First Printing - 1981

Compiled February 1, 1982

PAGE 47

In Section 4.3, BDOS File Operations,
Add two new BDOS Functions:



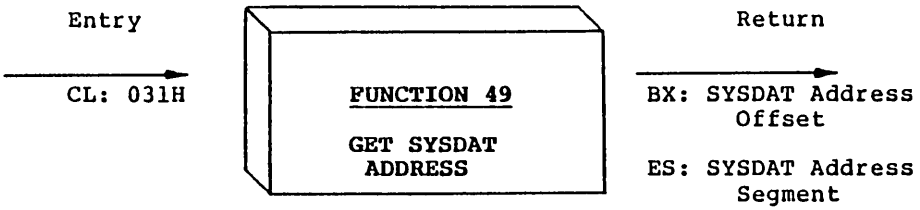
Load, Initialize, and Jump to specified Program

The CHAIN TO PROGRAM function provides a means of chaining from one program to the next without operator intervention. Although there is no passed parameter for this call, the calling process must place a command line terminated by a null byte in the default DMA buffer.

Under CP/M-86™, the CHAIN TO PROGRAM function releases the memory of the calling function before executing the command. The command line is parsed and placed in the Base Page of the new program. The Console Command Processor (CCP) then executes the command line.

PAGE 47 (continued)

Then, add:



Return the address of the System Data Area

The GET SYSDAT function returns the address of the System Data Area. The system data area includes the following information:

```
dmaad              equ     word ptr 0      ;user DMA address
dmabase            equ     word ptr 2      ;user DMA base
curdsk             equ     byte ptr 4      ;current user disk
usrcode            equ     byte ptr 5      ;current user number
control_p_flag     equ     byte ptr 22     ;listing toggle...
                                           ;set by ctrl-p
console_width      equ     byte ptr 64
printer_width      equ     byte ptr 65
console_column     equ     byte ptr 66
printer_column     equ     byte ptr 67
```

The following list provides an explanation of system data area parameters.

- dmaad means current user DMA address.
- dmabase means current user DMA base. (See page 48 under Function 51 in the CP/M-86 Operating System System Guide).
- curdsk means current user disk, 0-15 (A-P).
- usrcode means current user area, 0-15.
- control_p_flag, 0 means do not echo console output to the printer. FF means echo to the printer.

PAGE 60

In Table 5-4. BIOS Subroutine Summary, in the description of subroutine INIT,
change:

BDOS offset (0B11H)

to:

BDOS offset **(0B06H)**

CP/M-86™ Operating System

USER'S GUIDE

Adding your own text to the HELP.HLP file

CP/M-86™ is distributed with two related HELP files: HELP.CMD and HELP.HLP. The HELP.CMD file is the command file that processes the text of the HELP.HLP file and displays it on the screen. The HELP.HLP file is a text file to which you can add customized information, but you cannot edit the HELP.HLP file. You must use the HELP.CMD file to convert HELP.HLP to a file named HELP.DAT before you can edit or add your own text.

Use the following forms of the HELP command to change HELP.HLP to HELP.DAT and change HELP.DAT back to HELP.HLP.

```
HELP [E]

HELP [C]
```

The HELP [E] command accesses the file HELP.HLP on the default drive, removes the header record, and creates a file called HELP.DAT on the default drive. You can now invoke a word-processing program to edit or add your own text to the HELP.DAT file.

The HELP [C] command accesses your edited HELP.DAT file on the default drive, generates a new index for the entries record, and builds a revised HELP.HLP file on the default drive. HELP.CMD can now display your new HELP.HLP file.

You must add topics and subtopics to the HELP.DAT file in a specific format. The general format of a topic heading in the HELP.DAT file is shown below.

```
///nTOPICNAME<cr>
```

The three back slashes are the topic delimiters and must begin in column one. In the format statement above, n is a number in the range from 1 through 9 that signifies the level of the topic. A main topic always has a level number of 1. The first subtopic has a level number of 2. The next subtopic has a level number of 3, and so forth up to a maximum of nine levels. TOPICNAME is the name of your topic, and allows a maximum of twelve characters. The entire line is terminated with a carriage return.

Use the following guidelines to properly edit and insert text into the HELP.DAT file.

- Topics should be ordered in ascending alphabetical order.
- Subtopics should be ordered in ascending alphabetical order within their respective supertopic.
- Levels must be indicated by a number 1 - 9.

Some examples of topic and subtopic lines in the HELP.HLP file are shown below.

```
///1NEW UTILITY<cr>

///2COMMANDS<cr>

///3EXAMPLES<cr>
```

The first example shown above illustrates the format of a main topic line. The second example shows how to number the first subtopic of that main topic. The third example shows how the next level subtopic should be numbered. Any topicname with a level number of 1 is a main topic. Any topicname with a level number of 2 is a subtopic within its main topic.

When you are executing the HELP.CMD file, you need only enter enough letters of the topic to unambiguously identify the topic name. When referencing a subtopic, you must type the topic name AND the subtopic, otherwise the HELP program cannot determine which main topic you are referencing. You can also enter a topic and subtopic following the program's internal prompt, HELP>, as shown below.

```
HELP>ED COMMANDS
```

This form of HELP displays information about commands internal to the editing program, ED.

CP/M-86™ Operating System

SYSTEM GUIDE

"Diskette Track Buffering Greatly Increases Performance of the CP/M-86 Operating System"

by John R. Pierce

December 12, 1981

Addendum to the First Printing - 1981

Compiled February 1, 1982

Rotational latency is the major performance bottleneck in diskette systems. The standard eight-inch diskette rotates at only 360 RPM or 6 turns/second, and a read coming at a random time might take up to a full turn of the diskette or 167 milliseconds. Diskette-based operating systems often compensate for this by staggering track sectors, so several can be read in one turn. However, systems still require several turns to read all of the sectors of a particular track.

There are several techniques for reducing rotational latency. One of the simplest and most effective of these methods is track buffering; a track buffered system never needs more than two turns to read an entire track. Two turns require only a third of a second (worst case) instead of the full second or more required by the standard technique of reading the sectors out of order, according to a skew table traditionally used by CP/M® systems. In fact, 50% of the time, only 1.5 turns are necessary. This translates to an average of .167*1.5 seconds, or about a quarter second to read the track (which contains up to 8192 bytes in a double-density 8-inch floppy diskette).

However, nothing is free. Track buffering requires that the CBIOS contain a buffer large enough to hold the complete track, often 8192 bytes. Because most 8086 systems have plenty of memory, this should not cause a problem. Also, diskettes formatted with physically staggered sectors require multiple turns to read all sectors, resulting in significant performance degradation. This can only be remedied by copying these diskettes onto consecutively-skewed diskettes.

The following algorithm implements this track buffering scheme, in a fashion compatible with any existing CP/M diskette format. You must insert this module into your CBIOS, using the existing disk drivers to perform the TRACK_READ and SECTOR_WRITE functions. The EQUates for HOST_SECTSIZ, HOST_SPT, and HOST_FSN should be set to the appropriate values outlined in the comments.

A potential problem with any deblocking scheme is knowing when to "flush" the buffer following writes. The crudest scheme is to allow each write to cause an immediate disk write. This, however, takes a turn of the disk for each 128 byte record. Under CP/M, because all output files must be closed, and all closes cause a directory write, you can assume that you can save the records in memory, as long as you flush the buffer after each directory write. Conveniently, CP/M-86's BDOS sets a flag in CL when calling WRITE, indicating whether this is a write to the directory or not. This is the same scheme used by the standard sector blocking and deblocking algorithm distributed with CP/M-86™. The track buffering algorithm also notes which disk sectors have been updated in the buffer. When the algorithm writes from the buffer, it need only write to the updated physical disk sectors.

The TRACK_READ routine may consist of a loop that invokes your sector read for each sector. However, many disk controllers can read a whole track with a single command. Indeed, with some controllers, this is the only way to read a track in one turn. Optimization is also achieved by reading the track starting with the next sector passing under the heads. This method cuts the rotational latency to a fixed single turn rather than the one to two turns required if you must wait for sector one to start reading. Note that this possibility is highly controller-dependent, and generally requires a "read identification" capability to identify the next sector number. However, it should increase performance by about another 30%.

When using track buffering, the performance of a read-back check after each write causes much less degradation than when reading and writing individual sectors. This is because the check takes only one additional turn per track, rather than 26 or more. Furthermore, on a read-back check error, it would even be possible to re-write the bad sector in an attempt to correct it. This reduces the error rate for eight-inch diskettes from its present very low value to virtually none, while slowing writes down by only 30% or less.

Note that NO provision is made in this algorithm for handling diskette errors. It is assumed that the TRACK_READ and SECTOR_WRITE subroutines print appropriate error messages and perhaps even obtain operator responses. This is because an error may occur when writing a buffer, while CP/M thinks you are reading from the other drive! The only module that can handle disk errors properly is the BIOS itself.

If interrupts occur when the diskette door is opened, you can check the write flag to see if the buffer is dirty, and either clear the write flag and SEC_FLAGS array, or indicate that a write has occurred with a beep, or in some other fashion. If the system has programmable status lights, it is a good idea to set a light when WRITE_FLAG is set, and clear the light when the flag is cleared. If the system supports a programmable door lock mechanism, it can be set while the buffer is dirty, making the system failsafe.

These track buffering algorithms work with any sector size that is an integral multiple of 128, and not necessarily a power of two. This allows implementation of more dense diskette formats. Naturally, any system that implements nonstandard diskette formats should still have some way to read standard CP/M 3740 format diskettes for interchange.

The following is a Source Listing of the CP/M-86 Accelerator Track Buffering Routine for CP/M-86.

```
;       * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
;       *                                                           *
;       *    CP/M-86 Accelerator --  Track Buffering Routines       *
;       *                                                           *
;       *       This module, when installed in a CBIOS, causes      *
;       *       CP/M-86 to perform disk input output on a           *
;       *       track by track basis, rather than sector by         *
;       *       sector.                                             *
;       *                                                           *
;       *       This speeds diskette access, often by a             *
;       *       factor of four or more times.                       *
;       *                                                           *
;       *       The actual disk sectors must be an integral         *
;       *       multiple of 128 bytes, but do not need to be        *
;       *       a power of two multiple, unlike the deblocking      *
;       *       algorithms supplied with CP/M-86.                   *
;       *                                                           *
;       * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *


;  The following three equates must be set to correspond to the
; actual disk utilized.

host_sectsiz    equ     1024    ; bytes per actual (physical)
                                ; disk sector
host_spt        equ     8       ; actual sectors per track
host_fsn        equ     1       ; starting sector number
                                ; (only 0 or 1 allowed)
cpm_fsn         equ     0       ; first sector from CP/M

init:
        call    clear_flags             ; Initialize track buffering
         .
         .
         .
        jmp     CCP_entry


seldsk:
        mov cpm_disk,cl                 ; save the selected drive
        test dl,1                       ; check logged-in bit
        jnz old_disk                    ; not first time
```

```
                                        ; selected if nz

            ; here if CP/M is about to login to the drive being
            ; selected.

old_disk:
        mov bl,cpm_disk ! mov bh,0
        mov cl,4 ! shl bx,cl            ; times 16
        add bx,offset dpbase            ; gives offset from DPBASE
        ret                             ; back to BDOS

setdma:
        mov     dma_offset,cx           ; save DMA offset address
        ret
setdma_seg:
        mov     dma_segment,cx          ; save DMA segment address
        ret

home:
        test wr_flag,1 ! jnz homel ; if the buffer is clean,
        mov     cur_disk,-1     ; insure we read the directory
                                ; by invalidating
                                ; the track buffer
homel:
        mov     cx,0            ; home is a settrk zero

settrk:
        mov     cpm_track,cx    ; save track number for next operation
        ret
setsec:
        mov     cpm_sec,cx      ; save sector number
                                ; for next operation
        ret

sectran:
        mov     bx,cx           ; Put logical sector into dest. reg.
        test    dx,dx           ; see if table address is zero
        jz      sectran_exit    ; yeah, logical = physical
        add     bx,dx           ; else, we need to fetch the
        mov     bl,[BX]         ; actual sector number from the table
        mov     bh,0            ; zero high byte for good luck
sectran_exit:
        ret

read:
        call    setup
        push    es                      ; save the extra
                                        ; segment register
        mov     si,offset track_buffer  ; source segment
                                        ; is systems DS:
        add     si,ax                   ; gives the offset
                                        ; into the buffer
        les     di,dma_longword         ; point ES:DI at
                                        ; the users sector
        rep movsw                       ; doit
```

```
        pop     es                      ; restore the extra segment
        sub     ax,ax                   ; make a zero return code
        ret

write:
        push    cx                      ; save the write mode
                                        ; from the BDOS
        call    setup
        push    ax                      ; save buffer offset
        push    ds                      ; save the data segment
        push    es                      ; save the extra segment
        mov bx,ds ! mov es,bx           ; destination is our
                                        ; data segment
        mov     di,offset track_buffer  ; destination is in
                                        ; track buffer
        add     di,ax                   ; plus appropriate offset
        lds     si,dma_longword         ; source is users DMA address
        rep movsw                       ; move that sector
        pop     es                      ; restore the extra segment
        pop     ds                      ; and the data
                                        ; segment registers
        pop     ax                      ; recover buffer offset
        mov     cx,host_sectsiz         ; setup to divide by
                                        ; host sector size
        sub     dx,dx                   ; extend ax to 32 bits
        div     cx                      ; find out which host
                                        ; sector we changed
        mov     bx,ax                   ; put into index [BX]
        mov     sec_flags[BX],1         ; set the update flag
                                        ; for that sector
        mov     wr_flag,1               ; also set the dirty
                                        ; buffer flag
        pop     cx                      ; recover BDOS write code
        cmp     cl,1                    ; is this a directory update ?
        jne     return                  ; no, we may leave
                                        : dirty records in the buffer
        call    flush_buffer            ; we have a directory
                                        : write, need to
                                        ; flush the buffer
                                        : to insure the
                                        ; disks integrity
return:
        mov     ax,0                    ; never return BAD SECTOR code
        ret


setup:                  ; common code for setting up reads and writes

        mov     al,cpm_disk             ; see if selected disk is
        cmp     al,cur_disk             ;       the same as last time
        jne     wrong_track             ; no, we have wrong track

        mov     ax,cpm_track            ; see if desired track is
                                        ; same as
        cmp     ax,cur_track            ;       the track in the buffer
```

```
        je      correct_track           ; same drive and track,
                                        ; we don't need to read

;       Desired operation is on a different track than is in our
;       buffer, so it will be necessary to read in the desired track.
;       First, we must check to see if any sectors of the current
;       buffer are dirty.

wrong_track:
        call    flush_buffer            ; write any old records,
                                        ; if necessary

        mov     ax,cpm_track            ; get desired track number
        mov     cur_track,ax            ; make in new track
        mov     al,cpm_disk             ; get desired disk number
        mov     cur_disk,al             ; make it current drive
        mov     cur_dma,offset track_buffer ; point dma offset
                                        ; at track buffer
        mov     cur_sec,host_fsn        ; starting from first sector
        call    track_read              ; load the track

correct_track:
        mov     ax,cpm_sec              ; get the cp/m sector number
  if (cpm_fsn ne 0)
        sub     ax,cpm_fsn              ; correct if we start
                                        ; with sector one
    endif
        mov     cl,7                    ; log2(128)
        shl     ax,cl                   ; sector times 128
                                        ; gives offset
        mov cx,64 ! cld                 ; move 64 words forward
        ret


flush_buffer:
        test    wr_flag,1               ; see if we have anything
                                        ; to write
        jz      no_flush                ; no, skip scanning
                                        ; for dirty sectors
        mov     bx,0                    ; start at host sector 0
        mov     cx,host_spt             ; for host_spt sectors...
next_sect:
        test    sec_flags[BX],1         ; see if this sector
                                        ; has been changed
        jz      not_updated             ; no, leave it alone
        mov     sec_flags[BX],0         ; zero the flag for next time
        push    bx                      ; save the registers
        push    cx
        mov     ax,host_sectsiz
        mul     bx                      ; make track buffer offset
        add     ax,offset track_buffer  ; make direct pointer
        mov     cur_dma,ax              ; save for write routine
  if (host_fsn ne 0)
        add     bx,host_fsn
  endif
```

```
        mov     cur_sec,bx              ; save host sector number
        call    sector_write
        pop     cx
        pop     bx
not_updated:
        inc     bx
        loop    next_sect
no_flush:
        mov     wr_flag,0               ; clear the dirty buffer flag
        ret

clear_flags:    ; Clear all variables associated with the track
                ; buffer, so next operation will have to read a track.
                ; This is involves clearing all write flags and
                ; setting the old drive code to the invalid -1.

        mov cur_disk,-1                 ; insure initial pre-read
        sub ax,ax                       ; make a zero
        mov wr_flag,al                  ; clear the dirty buffer flag
        mov di,offset sec_flags         ; point to the update
                                        ; flag list
        mov bx,ds ! mov es,bx           ; ES <- DS
        mov cx,host_spt ! cld           ; set length and direction
        rep stosb                       ; zero the sector update flags
        ret

track_read:
                ; read an entire track from the drive "cur_disk",
                ;       the track "cur_track" into "track_buffer".

        ret

sector_write:
                ; write a physical sector to disk "cur_disk",
                ;       track "cur_track", sector "cur_sec" from
                ;       the buffer at DS:"cur_dma".

        ret


        dseg

cpm_disk        rb      1
cpm_track       rw      1
cpm_sec         rw      1

dma_offset      rw      1
dma_segment     rw      1
dma_longword    equ dword ptr dma_offset

cur_disk        rb      1
cur_sec         rw      1
cur_track       rw      1
cur_dma         rw      1
```

```
bdos_wr_code    rb      1       ; 1 indicates a directory write
wr_flag         rb      1       ; bit 0 on indicates we have a
                                ; dirty buffer

sec_flags       rb      host_spt ; bit 0 of each byte on indicates
                                 ;     corresponding host sector has
                                 ;     been updated and needs writing.

track_buffer    rb      host_sectsiz * host_spt
```

CP/M-86™ Operating System

PROGRAMMER'S GUIDE

Corrections to the First Printing - 1981

Compiled February 1982

Clarification of ASM-86™ Changes:

1) Forward references in EQU's are flagged as errors.

2) A ! in a comment is ignored; comments extend to the physical end of the line.

3) New directives: IFLIST and NOIFLIST control listing of false IF blocks.

4) IF directives can be nested to five levels.

5) New mnemonics implemented:

   - JC, JNC
   - CMPSB, CMPSW, LODSB, LODSW, MOVSB, MOVSW, SCASB, SCASW, STOSB, STOSW

6) JNBE implemented correctly.

7) Segment override prefix is allowed in source operand of string instructions.

8) Relational operators in expressions return 0FFFFH if true.

9) Abort if invalid command tail encountered.

10) Abort if symbol table overflows.

11) Abort if disk or directory full.

12) Incomplete string flagged as error (no terminating quote).

13) Error reported if an invalid numeric quantity appears in EQU directive.

14) Source files are opened in R/O mode for multiple access under MP/M-86™.

15) Format of .LST file:

- form-feed at start of file
- no form-feed at end of file
- no cr, lf at top of each page
- fewer lines per page
- spaces between hex bytes deleted to allow more space for comments
- errors printed when NOLIST active
- absolute address field for relative instructions

16) Format of .SYM file:

- form-feed at start of file
- symbols alphabetized within groups
- tabs expanded if symbols sent to printer ($SY)

17) Include files:

- filetype defaults to .A86
- filetype can have fewer than three characters
- abort if include file not found
- default to same drive as source when $a switch used

18) Programs with INCLUDE directives assemble correctly under CP/M® 1.4.

19) About 5.5K more space available for symbol table.

20) Use factor indicated at end of assembly (% usage of symbol table space).

21) Runs somewhat faster (especially with $PZ switch).

Clarification of DDT-86™ Changes:

1) User programs default to CCP stack, rather than local stack in DDT-86.

2) A command line starting with a ; is treated as a comment.

3) Interrupts are disabled while a single instruction is being traced.

4) BDOS error mode is set to return BDOS errors for MP/M-86.

5) Files are closed after reading and loading for MP/M-86.

6) New Block Compare function implemented, with the same command form as the Move function.

CP/M-86™ Operating System

Implementation Note

Notes for operation of CP/M-86 with the ISBC™ 86/12 and ISBC™ 204 Controller Boards

Compiled February 1982

The standard CP/M-86™ release is set up for operation with the Intel®SBC™ 86/12a and SBC™ 204 diskette controller, with two Shugart SA-800™ single density drives. The SBC 86/12 board has 32K bytes on board that is set up starting at location zero. Additional RAM is assumed to start at location 10000H (paragraph 1000H). The initial values of the segment table define this additional RAM area to be 64K bytes in length as provided in the BASIC I/O System (BIOS). Refer to the GETSEGT BIOS entry point, as well as the SEGTABLE data areas in the BIOS and CBIOS (listed in Appendixes D and E of the CP/M-86 Operating System System Guide) for the segment table definition.

Note that you can operate with less than 64K bytes of additional RAM (a 32K RAM area at 800H suffices), but the segment table must be changed before operating with programs which assume the full 64K is available. You can, for example, immediately enter DDT86 and manually alter the segment table in the BIOS to reflect the reduced memory configuration. Upon returning from DDT86 to the CCP level, any remaining transient programs,such as ED and ASM86, operate properly until the next cold start. Permanent segment table changes can be accomplished by editing the BIOS using this temporary CP/M-86 system or a CP/M-80™ system.

To use the distribution system, the SA-800, SBC 86/12a, and the SBC 204, boards must be "jumpered" in the following manner. See the Shugart and Intel hardware for the exact jumpering details.

The SA-800 Diskette Drive "A" is jumpered as follows:

Install Jumpers:
T1, T2, T3, T4, T5, T6, DS1, DC, 800, Z, A, B, C, DS

Remove Jumpers:
HL, DDS

Cut Trace:
RR

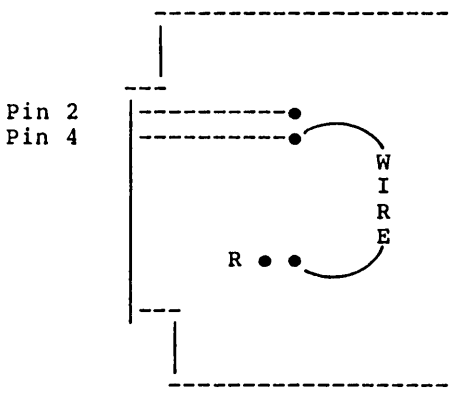
The SA-800 Diskette Drive "B" is jumpered as follows:

Install Jumpers:
T2, DS2, DC, 800, Z, A, B, C, DS

Remove Jumpers:
HL,DDS

Cut Traces:
R, RR

Wire a connection from wire wrap pin at edge connector pin 4 to wire wrap pin at right side of pair at "R" as shown below (only for drive "B"). This connection implements "Radial Ready."



The SBC 204 Diskette Controller is jumpered by installing the following connections:

Switches to Select Port A0 through AF:
1, 2, 3, 4, 6 and 8 are OFF
5 and 7 are ON

Install Jumpers:
55-56 (Serial Priority), 1-8, 19-20, 23-24,
26-27, 77-78, 75-76

The SBC 86/12a (or 86/12) CPU card is jumpered as follows:

Install Jumpers:
- 65 through 91: Interrupts as desired *
- 5-6 (Time-Out Acknowledge)
- 7 through 37: Parallel I/O as desired **
- 40-39, 43-42 (Baud Rate from PIC Channel 2)
- 54-55, 56-57, 59-60 (PIC Clocks)
- 92-93 (CPU Clock)
- 103-104, 105-106 (Bus Clocks from CPU)
- 151-152 (Serial Priority)
- 94-96, 97-98 (ROM's are 2716 Type)
- 127-128 (On-Board RAM is at 00000H)

Switches:
- 1, 2, and 8 are ON
- 3, 4, 5, 6, and 7 are OFF

Even ROM (0) in Socket A29
Odd ROM (1) in Socket A47

Notes:

* CP/M-86 does not use interrupts. Normally 65 through 91 are unchanged from the factory configuration.

** CP/M-86 does not use parallel I/O. Normally 7 through 37 remain unchanged.

CP/M-86™ V1.0, Application Note 01, 11/6/81

DDT-86™ SCREEN WIDTH ALTERATION

Applicable Products and Version Numbers: CP/M-86 V1.1, DDT-86

You can alter DDT-86 for use with 40 character wide consoles. The display of memory locations (D command) and the CPU state (X, T and U commands) reflect the narrower screen size. Make sure you have a back-up copy of DDT-86 before installing the patch as shown below.

```
A>ddt86
DDT86 1.1
-rddt86.cmd
START       END
nnnn:0000   nnnn:367F
-s12f0
nnnn:12F0 00 01
nnnn:12F1 00 .
-wddt86.cmd
-^c
A>
```

Licensed users are granted the right to include these changes in CP/M-86 V1.1 software. CP/M-86 and DDT-86 are trademarks of Digital Research.

SMALLER VERSIONS OF DDT-86™

Applicable Products and Version Numbers: CP/M-86 V1.0, DDT-86

You can create smaller versions of DDT-86 that may be useful for systems with limited memory. You can remove the assembler portion resulting in a 9K version of DDT-86 or you can remove both the assembler and disassembler resulting in a 5K version of DDT-86. In the 9K version, DDT-86 responds to an A command with a question mark. In the 5K version, both the A and L commands yield a question mark.

```
A>ddt86
DDT86 1.0
-rddt86.cmd
START       END
nnnn:0000 nnnn:367F
-s0
nnnn:0000 01
nnnn:0001 60 0d
nnnn:0002 03 02
nnnn:0003 00
nnnn:0004 00
nnnn:0005 66 0d
nnnn:0006 03 02
nnnn:0007 00.
-s1286
nnnn:1286 01 00
nnnn:1287 00 .
-wddt9k.cmd,0,217f
-^c
A>
```

Use the following procedure to remove the assembler and the disassembler from DDT-86.

```
A>ddt86
DDT86 1.0
-rddt86.cmd
START       END
nnnn:0000 nnnn:367F
-s0
nnnn:0000 01
nnnn:0001 60 2b
nnnn:0002 03 01
nnnn:0003 00
nnnn:0004 00
nnnn:0005 66 32
nnnn:0006 03 01
nnnn:0007 00.
```

```
-s1286
nnnn:1286 01 00
nnnn:1287 00 .
-s12b9
nnnn:12B9 01 00
nnnn:12BA 00 .
-wddt5k.cmd,0,13ff
-^c
A>
```

Licensed users are granted the right to include these changes in CP/M-86 V1.0 software. CP/M-86 and DDT-86 are trademarks of Digital Research.

BDOS DATA PAGE "TOD/DATA" FIELDS

Applicable products and version numbers: CP/M-86™ V1.1

Program: BDOS

The date field is located at the base of the data page + 32D bytes. The date field format is:

MM/DD/YY,

MM is the month (ASCII)
DD is the day (ASCII)
YY is the year (ASCII)

The time field is located at the base of the data page + 41D bytes. The time field format is:

HH:MM:SS,

HH is the hour (ASCII)
MM is the minute (ASCII)
SS is the second (ASCII)

The slash, colon and comma are literal characters in both the time and date representation.

These fields are initialized and displayed with the TOD command. (See the CP/M-86 Operating System User's Guide, pages 72-73.)