



Post Office Box 579, Pacific Grove, California 93950, (408) 649-3896

AN INTRODUCTION TO CP/M FEATURES AND FACILITIES

COPYRIGHT (c) 1976, 1977, 1978

DIGITAL RESEARCH

REVISION OF JANUARY 1978

Table of Contents

Section	Page
1. INTRODUCTION	1
2. FUNCTIONAL DESCRIPTION OF CP/M	3
2.1. General Command Structure	3
2.2. File References	3
3. SWITCHING DISKS	6
4. THE FORM OF BUILT-IN COMMANDS	7
4.1. ERA afn cr	7
4.2. DIR afn cr	8
4.3. REN ufn1=ufn2 cr	8
4.4. SAVE n ufn cr	9
4.5. TYPE ufn cr	9
5. LINE EDITING AND OUTPUT CONTROL.....	11
6. TRANSIENT COMMANDS	12
6.1. STAT cr	13
6.2. ASM ufn cr	16
6.3. LOAD ufn cr	17
6.4. PIP cr	18
6.5. ED ufn cr	25
6.6. SYSGEN cr	27
6.7. SUBMIT ufn parm#1 ... parm#n cr	28
6.8. DUMP ufn cr	30
6.9. MOVCPM cr	30
7. BDOS ERROR MESSAGES	33
8. OPERATION OF CP/M ON THE MDS	34

1. INTRODUCTION.

CP/M is a monitor control program for microcomputer system development which uses IBM-compatible flexible disks for backup storage. Using a computer mainframe based upon Intel's 8080 microcomputer, CP/M provides a general environment for program construction, storage, and editing, along with assembly and program check-out facilities. An important feature of CP/M is that it can be easily altered to execute with any computer configuration which uses an Intel 8080 (or Zilog Z-80) Central Processing Unit, and has at least 16K bytes of main memory with up to four IBM-compatible diskette drives. A detailed discussion of the modifications required for any particular hardware environment is given in the Digital Research document entitled "CP/M System Alteration Guide." Although the standard Digital Research version operates on a single-density Intel MDS 800, several different hardware manufacturers support their own input-output drivers for CP/M.

The CP/M monitor provides rapid access to programs through a comprehensive file management package. The file subsystem supports a named file structure, allowing dynamic allocation of file space as well as sequential and random file access. Using this file system, a large number of distinct programs can be stored in both source and machine executable form.

CP/M also supports a powerful context editor, Intel-compatible assembler, and debugger subsystems. Optional software includes a powerful Intel-compatible macro assembler, symbolic debugger, along with various high-level languages. When coupled with CP/M's Console Command Processor, the resulting facilities equal or excel similar large computer facilities.

CP/M is logically divided into several distinct parts:

BIOS	Basic I/O System (hardware dependent)
BDOS	Basic Disk Operating System
CCP	Console Command Processor
TPA	Transient Program Area

The BIOS provides the primitive operations necessary to access the diskette drives and to interface standard peripherals (teletype, CRT, Paper Tape Reader/Punch, and user-defined peripherals), and can be tailored by the user for any particular hardware environment by "patching" this portion of CP/M. The BDOS provides disk management by controlling one or more disk drives containing independent file directories. The BDOS implements disk allocation strategies which provide fully dynamic file construction while minimizing head movement across the disk during access. Any particular file may contain any number of records, not exceeding the size of any single disk. In a standard CP/M system, each disk can contain up to 64 distinct files. The

BDOS has entry points which include the following primitive operations which can be programmatically accessed:

SEARCH	Look for a particular disk file by name.
OPEN	Open a file for further operations.
CLOSE	Close a file after processing.
RENAME	Change the name of a particular file.
READ	Read a record from a particular file.
WRITE	Write a record onto the disk.
SELECT	Select a particular disk drive for further operations.

The CCP provides symbolic interface between the user's console and the remainder of the CP/M system. The CCP reads the console device and processes commands which include listing the file directory, printing the contents of files, and controlling the operation of transient programs, such as assemblers, editors, and debuggers. The standard commands which are available in the CCP are listed in a following section.

The last segment of CP/M is the area called the Transient Program Area (TPA). The TPA holds programs which are loaded from the disk under command of the CCP. During program editing, for example, the TPA holds the CP/M text editor machine code and data areas. Similarly, programs created under CP/M can be checked out by loading and executing these programs in the TPA.

It should be mentioned that any or all of the CP/M component subsystems can be "overlayed" by an executing program. That is, once a user's program is loaded into the TPA, the CCP, BDOS, and BIOS areas can be used as the program's data area. A "bootstrap" loader is programmatically accessible whenever the BIOS portion is not overlayed; thus, the user program need only branch to the bootstrap loader at the end of execution, and the complete CP/M monitor is reloaded from disk.

It should be reiterated that the CP/M operating system is partitioned into distinct modules, including the BIOS portion which defines the hardware environment in which CP/M is executing. Thus, the standard system can be easily modified to any non-standard environment by changing the peripheral drivers to handle the custom system.

2. FUNCTIONAL DESCRIPTION OF CP/M.

The user interacts with CP/M primarily through the CCP, which reads and interprets commands entered through the console. In general, the CCP addresses one of several disks which are online (the standard system addresses up to four different disk drives). These disk drives are labelled A, B, C, and D. A disk is "logged in" if the CCP is currently addressing the disk. In order to clearly indicate which disk is the currently logged disk, the CCP always prompts the operator with the disk name followed by the symbol ">" indicating that the CCP is ready for another command. Upon initial start up, the CP/M system is brought in from disk A, and the CCP displays the message

xxK CP/M VER m.m

where xx is the memory size (in kilobytes) which this CP/M system manages, and m.m is the CP/M version number. All CP/M systems are initially set to operate in a 16K memory space, but can be easily reconfigured to fit any memory size on the host system (see the MOVCPM transient command). Following system signon, CP/M automatically logs in disk A, prompts the user with the symbol "A>" (indicating that CP/M is currently addressing disk "A"), and waits for a command. The commands are implemented at two levels: built-in commands and transient commands.

2.1. GENERAL COMMAND STRUCTURE.

Built-in commands are a part of the CCP program itself, while transient commands are loaded into the TPA from disk and executed. The built-in commands are

ERA	Erase specified files.
DIR	List file names in the directory.
REN	Rename the specified file.
SAVE	Save memory contents in a file.
TYPE	Type the contents of a file on the logged disk.

Nearly all of the commands reference a particular file or group of files. The form of a file reference is specified below.

2.2. FILE REFERENCES.

A file reference identifies a particular file or group of files on a particular disk attached to CP/M. These file references can be either "unambiguous" (ufn) or "ambiguous" (afn). An unambiguous file reference uniquely identifies a single file, while an ambiguous file reference may be

satisfied by a number of different files.

File references consist of two parts: the primary name and the secondary name. Although the secondary name is optional, it usually is generic; that is, the secondary name "ASM," for example, is used to denote that the file is an assembly language source file, while the primary name distinguishes each particular source file. The two names are separated by a "." as shown below:

pppppppp.sss

where pppppppp represents the primary name of eight characters or less, and sss is the secondary name of no more than three characters. As mentioned above, the name

pppppppp

is also allowed and is equivalent to a secondary name consisting of three blanks. The characters used in specifying an unambiguous file reference cannot contain any of the special characters

< > . , ; : = ? * []

while all alphanumerics and remaining special characters are allowed.

An ambiguous file reference is used for directory search and pattern matching. The form of an ambiguous file reference is similar to an unambiguous reference, except the symbol "?" may be interspersed throughout the primary and secondary names. In various commands throughout CP/M, the "?" symbol matches any character of a file name in the "?" position. Thus, the ambiguous reference

X?Z.C?M

is satisfied by the unambiguous file names

XYZ.COM

and

X3Z.CAM

Note that the ambiguous reference

.

is equivalent to the ambiguous file reference

?????????.???

while

and pppppppp.*

*.sss

are abbreviations for

pppppppp.???

and

?????????.sss

respectively. As an example,

DIR *.*

is interpreted by the CCP as a command to list the names of all disk files in the directory, while

DIR X.Y

searches only for a file by the name X.Y Similarly, the command

DIR X?Y.C?M

causes a search for all (unambiguous) file names on the disk which satisfy this ambiguous reference.

The following file names are valid unambiguous file references:

X	XYZ	GAMMA
X.Y	XYZ.COM	GAMMA.1

As an added convenience, the programmer can generally specify the disk drive name along with the file name. In this case, the drive name is given as a letter A through Z followed by a colon (:). The specified drive is then "logged in" before the file operation occurs. Thus, the following are valid file names with disk name prefixes:

A:X.Y	B:XYZ	C:GAMMA
Z:XYZ.COM	B:X.A?M	C:*.ASM

It should also be noted that all alphabetic lower case letters in file and drive names are always translated to upper case when they are processed by the CCP.

3. SWITCHING DISKS.

The operator can switch the currently logged disk by typing the disk drive name (A, B, C, or D) followed by a colon (:) when the CCP is waiting for console input. Thus, the sequence of prompts and commands shown below might occur after the CP/M system is loaded from disk A:

16K CP/M VER 1.4

A>DIR List all files on disk A.

SAMPLE ASM

SAMPLE PRN

A>B: Switch to disk B.

B>DIR *.ASM List all "ASM" files on B.

DUMP ASM

FILES ASM

B>A: Switch back to A.

4. THE FORM OF BUILT-IN COMMANDS.

The file and device reference forms described above can now be used to fully specify the structure of the built-in commands. In the description below, assume the following abbreviations:

ufn	-	unambiguous file reference
afn	-	ambiguous file reference
cr	-	carriage return

Further, recall that the CCP always translates lower case characters to upper case characters internally. Thus, lower case alphabetic characters are treated as if they are upper case in command names and file references.

4.1 ERA afn cr

The ERA (erase) command removes files from the currently logged-in disk (i.e., the disk name currently prompted by CP/M preceding the ">"). The files which are erased are those which satisfy the ambiguous file reference afn. The following examples illustrate the use of ERA:

ERA X.Y	The file named X.Y on the currently logged disk is removed from the disk directory, and the space is returned.
ERA X.*	All files with primary name X are removed from the current disk.
ERA *.ASM	All files with secondary name ASM are removed from the current disk.
ERA X?Y.C?M	All files on the current disk which satisfy the ambiguous reference X?Y.C?M are deleted.
ERA *.*	Erase all files on the current disk (in this case the CCP prompts the console with the message "ALL FILES (Y/N)?" which requires a Y response before files are actually removed).
ERA B:*.PRN	All files on drive B which satisfy the ambiguous reference ???????.PRN are deleted, independently of the currently logged disk.

4.2. DIR afn cr

The DIR (directory) command causes the names of all files which satisfy the ambiguous file name afn to be listed at the console device. As a special case, the command

DIR

lists the files on the currently logged disk (the command "DIR" is equivalent to the command "DIR *.*"). Valid DIR commands are shown below.

DIR X.Y

DIR X?Z.C?M

DIR ??_Y

Similar to other CCP commands, the afn can be preceded by a drive name. The following DIR commands cause the selected drive to be addressed before the directory search takes place.

DIR B:

DIR B:X.Y

DIR B:*.A?M

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.

4.3. REN ufn1=ufn2 cr

The REN (rename) command allows the user to change the names of files on disk. The file satisfying ufn2 is changed to ufn1. The currently logged disk is assumed to contain the file to rename (ufn1). The CCP also allows the user to type a left-directed arrow instead of the equal sign, if the user's console supports this graphic character. Examples of the REN command are

REN X.Y=Q.R The file Q.R is changed to X.Y.

REN XYZ.COM=XYZ.XXX The file XYZ.XXX is changed to XYZ.COM.

The operator can precede either ufn1 or ufn2 (or both) by an optional drive address. Given that ufn1 is preceded by a drive name, then ufn2 is assumed to exist on the same drive as ufn1. Similarly, if ufn2 is preceded by a drive name, then ufn1 is assumed to reside on that drive as well. If both ufn1 and ufn2 are preceded by drive names, then the same drive must be

specified in both cases. The following REN commands illustrate this format.

REN A:X.ASM = Y.ASM	The file Y.ASM is changed to X.ASM on drive A.
REN B:ZAP.BAS=ZOT.BAS	The file ZOT.BAS is changed to ZAP.BAS on drive B.
REN B:A.ASM = B:A.BAK	The file A.BAK is renamed to A.ASM on drive B.

If the file ufn1 is already present, the REN command will respond with the error "FILE EXISTS" and not perform the change. If ufn2 does not exist on the specified diskette, then the message "NOT FOUND" is printed at the console.

4.4. SAVE n ufn cr

The SAVE command places n pages (256-byte blocks) onto disk from the TPA and names this file ufn. In the CP/M distribution system, the TPA starts at 100H (hexadecimal), which is the second page of memory. Thus, if the user's program occupies the area from 100H through 2FFH, the SAVE command must specify 2 pages of memory. The machine code file can be subsequently loaded and executed. Examples are:

SAVE 3 X.COM	Copies 100H through 3FFH to X.COM.
SAVE 40 Q	Copies 100H through 28FFH to Q (note that 28 is the page count in 28FFH, and that 28H = 2*16+8 = 40 decimal).
SAVE 4 X.Y	Copies 100H through 4FFH to X.Y.

The SAVE command can also specify a disk drive in the afn portion of the command, as shown below.

SAVE 10 B:ZOT.COM	Copies 10 pages (100H through 0AFFH) to the file ZOT.COM on drive B.
-------------------	--

4.5. TYPE ufn cr

The TYPE command displays the contents of the ASCII source file ufn on the currently logged disk at the console device. Valid TYPE commands are

TYPE X.Y

TYPE X.PLM

TYPE XXX

The TYPE command expands tabs (clt-I characters), assuming tab positions are set at every eighth column. The ufn can also reference a drive name as shown below.

TYPE B:X.PRN

The file X.PRN from drive B is displayed.

5. LINE EDITING AND OUTPUT CONTROL.

The CCP allows certain line editing functions while typing command lines.

rubout	Delete and echo the last character typed at the console.
ctl-U	Delete the entire line typed at the console.
ctl-X	(Same as ctl-U)
ctl-R	Retype current command line: types a "clean line" following character deletion with rubouts.
ctl-E	Physical end of line: carriage is returned, but line is not sent until the carriage return key is depressed.
ctl-C	CP/M system reboot (warm start)
ctl-Z	End input from the console (used in PIP and ED).

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

ctl-P	Copy all subsequent console output to the currently assigned list device (see the STAT command). 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.

Note that the ctl-key sequences shown above are obtained by depressing the control and letter keys simultaneously. Further, CCP command lines can generally be up to 255 characters in length; they are not acted upon until the carriage return key is typed.

6. TRANSIENT COMMANDS.

Transient commands are loaded from the currently logged disk and executed in the TPA. The transient commands defined for execution under the CCP are shown below. Additional functions can easily be defined by the user (see the LOAD command definition).

STAT	List the number of bytes of storage remaining on the currently logged disk, provide statistical information about particular files, and display or alter device assignment.
ASM	Load the CP/M assembler and assemble the specified program from disk.
LOAD	Load the file in Intel "hex" machine code format and produce a file in machine executable form which can be loaded into the TPA (this loaded program becomes a new command under the CCP).
DDT	Load the CP/M debugger into TPA and start execution.
PIP	Load the Peripheral Interchange Program for subsequent disk file and peripheral transfer operations.
ED	Load and execute the CP/M text editor program.
SYSGEN	Create a new CP/M system diskette.
SUBMIT	Submit a file of commands for batch processing.
DUMP	Dump the contents of a file in hex.
MOVCPM	Regenerate the CP/M system for a particular memory size.

Transient commands are specified in the same manner as built-in commands, and additional commands can be easily defined by the user. As an added convenience, the transient command can be preceded by a drive name, which causes the transient to be loaded from the specified drive into the TPA for execution. Thus, the command

B:STAT

causes CP/M to temporarily "log in" drive B for the source of the STAT transient, and then return to the original logged disk for subsequent processing.

The basic transient commands are listed in detail below.

6.1. STAT cr

The STAT command provides general statistical information about file storage and device assignment. It is initiated by typing one of the following forms:

```
STAT cr
STAT "command line" cr
```

Special forms of the "command line" allow the current device assignment to be examined and altered as well. The various command lines which can be specified are shown below, with an explanation of each form shown to the right.

STAT cr If the user types an empty command line, the STAT transient calculates the storage remaining on all active drives, and prints a message

```
                              x: R/W, SPACE: nnnK
                              or
                              x: R/O, SPACE: nnnK
```

for each active drive x, where R/W indicates the drive may be read or written, and R/O indicates the drive is read only (a drive becomes R/O by explicitly setting it to read only, as shown below, or by inadvertently changing diskettes without performing a warm start). The space remaining on the diskette in drive x is given in kilobytes by nnn.

STAT x: cr If a drive name is given, then the drive is selected before the storage is computed. Thus, the command "STAT B:" could be issued while logged into drive A, resulting in the message

```
                              BYTES REMAINING ON B: nnnK
```

STAT afn cr The command line can also specify a set of files to be scanned by STAT. The files which satisfy afn are listed in alphabetical order, with storage requirements for each file under the heading

```
                              RECS BYTS EX D:FILENAME.TYP
                              rrrr bbbK ee d:ppppppppp.sss
```

where rrrr is the number of 128-byte records

allocated to the file, bbb is the number of kilobytes allocated to the file ($bbb = rrrr * 128 / 1024$), ee is the number of 16K extensions ($ee = bbb / 16$), d is the drive name containing the file (A...Z), pppppppp is the (up to) eight-character primary file name, and sss is the (up to) three-character secondary name. After listing the individual files, the storage usage is summarized.

STAT x:afn cr

As a convenience, the drive name can be given ahead of the afn. In this case, the specified drive is first selected, and the form "STAT afn" is executed.

STAT x:=R/O cr

This form sets the drive given by x to read-only, which remains in effect until the next warm or cold start takes place. When a disk is read-only, the message

BDOS ERR ON x: READ ONLY

will appear if there is an attempt to write to the read-only disk x. CP/M waits until a key is depressed before performing an automatic warm start (at which time the disk becomes R/W).

The STAT command also allows control over the physical to logical device assignment (see the IOBYTE function described in the manuals "CP/M Interface Guide" and "CP/M System Alteration Guide"). In general, there are four logical peripheral devices which are, at any particular instant, each assigned to one of several physical peripheral devices. The four logical devices are named:

CON:	The system console device (used by CCP for communication with the operator)
RDR:	The paper tape reader device
PUN:	The paper tape punch device
LST:	The output list device

The actual devices attached to any particular computer system are driven by subroutines in the BIOS portion of CP/M. Thus, the logical RDR: device, for example, could actually be a high speed reader, Teletype reader, or cassette tape. In order to allow some flexibility in device naming and assignment, several physical devices are defined, as shown below:

TTY:	Teletype device (slow speed console)
CRT:	Cathode ray tube device (high speed console)
BAT:	Batch processing (console is current RDR:, output goes to current LST: device)
UC1:	User-defined console
PTR:	Paper tape reader (high speed reader)
UR1:	User-defined reader #1
UR2:	User-defined reader #2
PTP:	Paper tape punch (high speed punch)
UP1:	User-defined punch #1
UP2:	User-defined punch #2
LPT:	Line printer
UL1:	User-defined list device #1

It must be emphasized that the physical device names may or may not actually correspond to devices which the names imply. That is, the PTP: device may be implemented as a cassette write operation, if the user wishes. The exact correspondence and driving subroutine is defined in the BIOS portion of CP/M. In the standard distribution version of CP/M, these devices correspond to their names on the MDS 800 development system.

The possible logical to physical device assignments can be displayed by typing

```
STAT VAL: cr
```

The STAT prints the possible values which can be taken on for each logical device:

```
CON: = TTY:  CRT:  BAT:  UC1:
RDR: = TTY:  PTR:  UR1:  UR2:
PUN: = TTY:  PTP:  UP1:  UP2:
LST: = TTY:  CRT:  LPT:  UL1:
```

In each case, the logical device shown to the left can take any of the four physical assignments shown to the right on each line. The current logical to physical mapping is displayed by typing the command

```
STAT DEV: cr
```

which produces a listing of each logical device to the left, and the current corresponding physical device to the right. For example, the list might appear as follows:

```
CON: = CRT:
RDR: = URl:
PUN: = PTP:
LST: = TTY:
```

The current logical to physical device assignment can be changed by typing a STAT command of the form

```
STAT ld1 = pd1, ld2 = pd2 , ... , ldn = pdn cr
```

where ld1 through ldn are logical device names, and pd1 through pdn are compatible physical device names (i.e., ldi and pdi appear on the same line in the "VAL:" command shown above). The following are valid STAT commands which change the current logical to physical device assignments:

```
STAT CON:=CRT: cr
STAT PUN: = TTY:,LST:=LPT:, RDR:=TTY: cr
```

6.2. ASM ufn cr

The ASM command loads and executes the CP/M 8080 assembler. The ufn specifies a source file containing assembly language statements where the secondary name is assumed to be ASM, and thus is not specified. The following ASM commands are valid:

```
ASM X
```

```
ASM GAMMA
```

The two-pass assembler is automatically executed. If assembly errors occur during the second pass, the errors are printed at the console.

The assembler produces a file

```
x.PRN
```

where x is the primary name specified in the ASM command. The PRN file contains a listing of the source program (with imbedded tab characters if present in the source program), along with the machine code generated for each statement and diagnostic error messages, if any. The PRN file can be listed

at the console using the TYPE command, or sent to a peripheral device using PIP (see the PIP command structure below). Note also that the PRN file contains the original source program, augmented by miscellaneous assembly information in the leftmost 16 columns (program addresses and hexadecimal machine code, for example). Thus, the PRN file can serve as a backup for the original source file: if the source file is accidentally removed or destroyed, the PRN file can be edited (see the ED operator's guide) by removing the leftmost 16 characters of each line (this can be done by issuing a single editor "macro" command). The resulting file is identical to the original source file and can be renamed (REN) from PRN to ASM for subsequent editing and assembly. The file

x.HEX

is also produced which contains 8080 machine language in Intel "hex" format suitable for subsequent loading and execution (see the LOAD command). For complete details of CP/M's assembly language program, see the "CP/M Assembler Language (ASM) User's Guide."

Similar to other transient commands, the source file for assembly can be taken from an alternate disk by prefixing the assembly language file name by a disk drive name. Thus, the command

ASM B:ALPHA cr

loads the assembler from the currently logged drive and operates upon the source program ALPHA.ASM on drive B. The HEX and PRN files are also placed on drive B in this case.

6.3. LOAD ufn cr

The LOAD command reads the file ufn, which is assumed to contain "hex" format machine code, and produces a memory image file which can be subsequently executed. The file name ufn is assumed to be of the form

x.HEX

and thus only the name x need be specified in the command. The LOAD command creates a file named

x.COM

which marks it as containing machine executable code. The file is actually loaded into memory and executed when the user types the file name x immediately after the prompting character ">" printed by the CCP.

In general, the CCP reads the name x following the prompting character and looks for a built-in function name. If no function name is found, the CCP searches the system disk directory for a file by the name

x.COM

If found, the machine code is loaded into the TPA, and the program executes. Thus, the user need only LOAD a hex file once; it can be subsequently executed any number of times by simply typing the primary name. In this way, the user can "invent" new commands in the CCP. (Initialized disks contain the transient commands as COM files, which can be deleted at the user's option.) The operation can take place on an alternate drive if the file name is prefixed by a drive name. Thus,

LOAD B:BETA

brings the LOAD program into the TPA from the currently logged disk and operates upon drive B after execution begins.

It must be noted that the BETA.HEX file must contain valid Intel format hexadecimal machine code records (as produced by the ASM program, for example) which begin at 100H, the beginning of the TPA. Further, the addresses in the hex records must be in ascending order; gaps in unfilled memory regions are filled with zeroes by the LOAD command as the hex records are read. Thus, LOAD must be used only for creating CP/M standard "COM" files which operate in the TPA. Programs which occupy regions of memory other than the TPA can be loaded under DDT.

6.4. PIP cr

PIP is the CP/M Peripheral Interchange Program which implements the basic media conversion operations necessary to load, print, punch, copy, and combine disk files. The PIP program is initiated by typing one of the following forms

- (1) PIP cr
- (2) PIP "command line" cr

In both cases, PIP is loaded into the TPA and executed. In case (1), PIP reads command lines directly from the console, prompted with the "*" character, until an empty command line is typed (i.e., a single carriage return is issued by the operator). Each successive command line causes some media conversion to take place according to the rules shown below. Form (2) of the PIP command is equivalent to the first, except that the single command line given with the PIP command is automatically executed, and PIP terminates immediately with no further prompting of the console for input command lines. The form of each command line is

destination = source#1, source#2, ... , source#n cr

where "destination" is the file or peripheral device to receive the data, and

"source#1, ..., source#n" represents a series of one or more files or devices which are copied from left to right to the destination.

When multiple files are given in the command line (i.e., $n > 1$), the individual files are assumed to contain ASCII characters, with an assumed CP/M end-of-file character (ctl-Z) at the end of each file (see the O parameter to override this assumption). The equal symbol (=) can be replaced by a left-oriented arrow, if your console supports this ASCII character, to improve readability. Lower case ASCII alphabets are internally translated to upper case to be consistent with CP/M file and device name conventions. Finally, the total command line length cannot exceed 255 characters (ctl-E can be used to force a physical carriage return for lines which exceed the console width).

The destination and source elements can be unambiguous references to CP/M source files, with or without a preceding disk drive name. That is, any file can be referenced with a preceding drive name (A:, B:, C:, or D:) which defines the particular drive where the file may be obtained or stored. When the drive name is not included, the currently logged disk is assumed. Further, the destination file can also appear as one or more of the source files, in which case the source file is not altered until the entire concatenation is complete. If the destination file already exists, it is removed if the command line is properly formed (it is not removed if an error condition arises). The following command lines (with explanations to the right) are valid as input to PIP:

X = Y cr

Copy to file X from file Y, where X and Y are unambiguous file names; Y remains unchanged.

X = Y,Z cr

Concatenate files Y and Z and copy to file X, with Y and Z unchanged.

X.ASM=Y.ASM,Z.ASM,FIN.ASM cr

Create the file X.ASM from the concatenation of the Y, Z, and FIN files with type ASM.

NEW.ZOT = B:OLD.ZAP cr

Move a copy of OLD.ZAP from drive B to the currently logged disk; name the file NEW.ZOT.

B:A.U = B:B.V,A:C.W,D:X cr

Concatenate file B.V from drive B with C.W from drive A and D.X. from the logged disk; create the file A.U on drive B.

For more convenient use, PIP allows abbreviated commands for transferring files between disk drives. The abbreviated forms are

PIP x:=afn cr

PIP x:=y:afn cr

PIP ufn = y: cr

PIP x:ufn = y: cr

The first form copies all files from the currently logged disk which satisfy the afn to the same file names on drive x (x = A...Z). The second form is equivalent to the first, where the source for the copy is drive y (y = A...Z). The third form is equivalent to the command "PIP ufn=y:ufn cr" which copies the file given by ufn from drive y to the file ufn on drive x. The fourth form is equivalent to the third, where the source disk is explicitly given by y.

Note that the source and destination disks must be different in all of these cases. If an afn is specified, PIP lists each ufn which satisfies the afn as it is being copied. If a file exists by the same name as the destination file, it is removed upon successful completion of the copy, and replaced by the copied file.

The following PIP commands give examples of valid disk-to-disk copy operations:

B:=*.COM cr	Copy all files which have the secondary name "COM" to drive B from the current drive.
A:=B:ZAP.* cr	Copy all files which have the primary name "ZAP" to drive A from drive B.
ZAP.ASM=B: cr	Equivalent to ZAP.ASM=B:ZAP.ASM
B:ZOT.COM=A: cr	Equivalent to B:ZOT.COM=A:ZOT.COM
B:=GAMMA.BAS cr	Same as B:GAMMA.BAS=GAMMA.BAS
B:=A:GAMMA.BAS cr	Same as B:GAMMA.BAS=A:GAMMA.BAS

PIP also allows reference to physical and logical devices which are attached to the CP/M system. The device names are the same as given under the STAT command, along with a number of specially named devices. The logical devices given in the STAT command are

CON: (console), RDR: (reader), PUN: (punch), and LST: (list)

while the physical devices are

TTY: (console, reader, punch, or list)
 CRT: (console, or list), UCl: (console)
 PTR: (reader), URl: (reader), UR2: (reader)
 PTP: (punch), UP1: (punch), UP2: (punch)
 LPT: (list), ULl: (list)

(Note that the "BAT:" physical device is not included, since this assignment is used only to indicate that the RDR: and LST: devices are to be used for console input/output.)

The RDR, LST, PUN, and CON devices are all defined within the BIOS portion of CP/M, and thus are easily altered for any particular I/O system. (The current physical device mapping is defined by IOBYTE; see the "CP/M Interface Guide" for a discussion of this function). The destination device must be capable of receiving data (i.e., data cannot be sent to the punch), and the source devices must be capable of generating data (i.e., the LST: device cannot be read).

The additional device names which can be used in PIP commands are

NUL: Send 40 "nulls" (ASCII 0's) to the device
 (this can be issued at the end of punched output).

EOF: Send a CP/M end-of-file (ASCII ctl-Z) to the
 destination device (sent automatically at the
 end of all ASCII data transfers through PIP).

INP: Special PIP input source which can be "patched"
 into the PIP program itself: PIP gets the input
 data character-by-character by CALLing location
 103H, with data returned in location 109H (parity
 bit must be zero).

OUT: Special PIP output destination which can be
 patched into the PIP program: PIP CALLs location
 106H with data in register C for each character
 to transmit. Note that locations 109H through
 1FFH of the PIP memory image are not used and
 can be replaced by special purpose drivers using
 DDT (see the DDT operator's manual).

PRN: Same as LST:, except that tabs are expanded at
 every eighth character position, lines are
 numbered, and page ejects are inserted every 60
 lines, with an initial eject (same as [t8np]).

File and device names can be interspersed in the PIP commands. In each case, the specific device is read until end-of-file (ctl-Z for ASCII files, and a real end of file for non-ASCII disk files). Data from each device or file is concatenated from left to right until the last data source has been

read. The destination device or file is written using the data from the source files, and an end-of-file character (ctl-Z) is appended to the result for ASCII files. Note if the destination is a disk file, then a temporary file is created (\$\$\$ secondary name) which is changed to the actual file name only upon successful completion of the copy. Files with the extension "COM" are always assumed to be non-ASCII.

The copy operation can be aborted at any time by depressing any key on the keyboard (a rubout suffices). PIP will respond with the message "ABORTED" to indicate that the operation was not completed. Note that if any operation is aborted, or if an error occurs during processing, PIP removes any pending commands which were set up while using the SUBMIT command.

It should also be noted that PIP performs a special function if the destination is a disk file with type "HEX" (an Intel hex formatted machine code file), and the source is an external peripheral device, such as a paper tape reader. In this case, the PIP program checks to ensure that the source file contains a properly formed hex file, with legal hexadecimal values and checksum records. When an invalid input record is found, PIP reports an error message at the console and waits for corrective action. It is usually sufficient to open the reader and rerun a section of the tape (pull the tape back about 20 inches). When the tape is ready for the re-read, type a single carriage return at the console, and PIP will attempt another read. If the tape position cannot be properly read, simply continue the read (by typing a return following the error message), and enter the record manually with the ED program after the disk file is constructed. For convenience, PIP allows the end-of-file to be entered from the console if the source file is a RDR: device. In this case, the PIP program reads the device and monitors the keyboard. If ctl-Z is typed at the keyboard, then the read operation is terminated normally.

Valid PIP commands are shown below.

PIP LST: = X.PRN cr	Copy X.PRN to the LST device and terminate the PIP program.
PIP cr	Start PIP for a sequence of commands (PIP prompts with "**").
*CON:=X.ASM,Y.ASM,Z.ASM cr	Concatenate three ASM files and copy to the CON device.
*X.HEX=CON:;Y.HEX,PTR: cr	Create a HEX file by reading the CON (until a ctl-Z is typed), followed by data from Y.HEX, followed by data from PTR until a ctl-Z is encountered.
*cr	Single carriage return stops PIP.

<pre>PIP PUN:=NUL:,X.ASM,EOF:,NUL: cr</pre>	Send 40 nulls to the punch device; then copy the X.ASM file to the punch, followed by an end-of-file (ctl-Z) and 40 more null characters.
---	---

The user can also specify one or more PIP parameters, enclosed in left and right square brackets, separated by zero or more blanks. Each parameter affects the copy operation, and the enclosed list of parameters must immediately follow the affected file or device. Generally, each parameter can be followed by an optional decimal integer value (the S and Q parameters are exceptions). The valid PIP parameters are listed below.

- B Block mode transfer: data is buffered by PIP until an ASCII x-off character (ctl-S) is received from the source device. This allows transfer of data to a disk file from a continuous reading device, such as a cassette reader. Upon receipt of the x-off, PIP clears the disk buffers and returns for more input data. The amount of data which can be buffered is dependent upon the memory size of the host system (PIP will issue an error message if the buffers overflow).
- Dn Delete characters which extend past column n in the transfer of data to the destination from the character source. This parameter is used most often to truncate long lines which are sent to a (narrow) printer or console device.
- E Echo all transfer operations to the console as they are being performed.
- F Filter form feeds from the file. All imbedded form feeds are removed. The P parameter can be used simultaneously to insert new form feeds.
- H Hex data transfer: all data is checked for proper Intel hex file format. Non-essential characters between hex records are removed during the copy operation. The console will be prompted for corrective action in case errors occur.
- I Ignore ":00" records in the transfer of Intel hex format file (the I parameter automatically sets the H parameter).
- L Translate upper case alphabetic to lower case.
- N Add line numbers to each line transferred to the destination starting at one, and incrementing by 1. Leading zeroes are suppressed, and the number is followed by a colon. If N2 is specified, then leading zeroes are included, and a tab is inserted following the number. The tab is expanded if T is

set.

- O Object file (non-ASCII) transfer: the normal CP/M end of file is ignored.
- Pn Include page ejects at every n lines (with an initial page eject). If n = 1 or is excluded altogether, page ejects occur every 60 lines. If the F parameter is used, form feed suppression takes place before the new page ejects are inserted.
- Qs[†]z Quit copying from the source device or file when the string s (terminated by ctl-Z) is encountered.
- Ss[†]z Start copying from the source device when the string s is encountered (terminated by ctl-Z). The S and Q parameters can be used to "abstract" a particular section of a file (such as a subroutine). The start and quit strings are always included in the copy operation.

NOTE - the strings following the s and q parameters are translated to upper case by the CCP if form (2) of the PIP command is used. Form (1) of the PIP invocation, however, does not perform the automatic upper case translation.

- (1) PIP cr
- (2) PIP "command line" cr

- Tn Expand tabs (ctl-I characters) to every nth column during the transfer of characters to the destination from the source.
- U Translate lower case alphabets to upper case during the the copy operation.
- V Verify that data has been copied correctly by rereading after the write operation (the destination must be a disk file).
- Z Zero the parity bit on input for each ASCII character.

The following are valid PIP commands which specify parameters in the file transfer:

- | | |
|-------------------------|--|
| PIP X.ASM=B:[v] cr | Copy X.ASM from drive B to the current drive and verify that the data was properly copied. |
| PIP LPT:=X.ASM[nt8u] cr | Copy X.ASM to the LPT: device; number each line, expand tabs to every eighth column, and translate lower case alphabets to upper case. |

PIP PUN:=X.HEX[i],Y.ZOT[h] cr First copy X.HEX to the PUN: device and ignore the trailing ":00" record in X.HEX; then continue the transfer of data by reading Y.ZOT, which contains hex records, including any ":00" records which it contains.

PIP X.LIB = Y.ASM [sSUBRl:↑z qJMP L3↑z] cr Copy from the file Y.ASM into the file X.LIB. Start the copy when the string "SUBRl:" has been found, and quit copying after the string "JMP L3" is encountered.

PIP PRN:=X.ASM[p50] Send X.ASM to the LST: device, with line numbers, tabs expanded to every eighth column, and page ejects at every 50th line. Note that nt8p60 is the assumed parameter list for a PRN file; p50 overrides the default value.

6.5. ED ufn cr

The ED program is the CP/M system context editor, which allows creation and alteration of ASCII files in the CP/M environment. Complete details of operation are given the ED user's manual, "ED: a Context Editor for the CP/M Disk System." In general, ED allows the operator to create and operate upon source files which are organized as a sequence of ASCII characters, separated by end-of-line characters (a carriage-return line-feed sequence). There is no practical restriction on line length (no single line can exceed the size of the working memory), which is instead defined by the number of characters typed between cr's. The ED program has a number of commands for character string searching, replacement, and insertion, which are useful in the creation and correction of programs or text files under CP/M. Although the CP/M has a limited memory work space area (approximately 5000 characters in a 16K CP/M system), the file size which can be edited is not limited, since data is easily "paged" through this work area.

Upon initiation, ED creates the specified source file, if it does not exist, and opens the file for access. The programmer then "appends" data from the source file into the work area, if the source file already exists (see the A command), for editing. The appended data can then be displayed, altered, and written from the work area back to the disk (see the W command). Particular points in the program can be automatically paged and located by context (see the N command), allowing easy access to particular portions of a large file.

Given that the operator has typed

ED X.ASM cr

the ED program creates an intermediate work file with the name

X.\$\$\$

to hold the edited data during the ED run. Upon completion of ED, the X.ASM file (original file) is renamed to X.BAK, and the edited work file is renamed to X.ASM. Thus, the X.BAK file contains the original (unedited) file, and the X.ASM file contains the newly edited file. The operator can always return to the previous version of a file by removing the most recent version, and renaming the previous version. Suppose, for example, that the current X.ASM file was improperly edited; the sequence of CCP command shown below would reclaim the backup file.

DIR X.*	Check to see that BAK file is available.
ERA X.ASM	Erase most recent version.
REN X.ASM=X.BAK	Rename the BAK file to ASM.

Note that the operator can abort the edit at any point (reboot, power failure, ctl-C, or Q command) without destroying the original file. In this case, the BAK file is not created, and the original file is always intact.

The ED program also allows the user to "ping-pong" the source and create backup files between two disks. The form of the ED command in this case is

ED ufn d:

where ufn is the name of a file to edit on the currently logged disk, and d is the name of an alternate drive. The ED program reads and processes the source file, and writes the new file to drive d, using the name ufn. Upon completion of processing, the original file becomes the backup file. Thus, if the operator is addressing disk A, the following command is valid:

ED X.ASM B:

which edits the file X.ASM on drive A, creating the new file X.\$\$\$ on drive B. Upon completion of a successful edit, A:X.ASM is renamed to A:X.BAK, and B:X.\$\$\$ is renamed to B:X.ASM. For user convenience, the currently logged disk becomes drive B at the end of the edit. Note that if a file by the name B:X.ASM exists before the editing begins, the message

FILE EXISTS

is printed at the console as a precaution against accidentally destroying a source file. In this case, the operator must first ERASE the existing file and then restart the edit operation.

Similar to other transient commands, editing can take place on a drive different from the currently logged disk by preceding the source file name by a drive name. Examples of valid edit requests are shown below

ED A:X.ASM	Edit the file X.ASM on drive A, with new file and backup on drive A.
ED B:X.ASM A:	Edit the file X.ASM on drive B to the temporary file X.\$\$\$ on drive A. On termination of editing, change X.ASM on drive B to X.BAK, and change X.\$\$\$ on drive A to X.ASM.

6.6. SYSGEN cr

The SYSGEN transient command allows generation of an initialized diskette containing the CP/M operating system. The SYSGEN program prompts the console for commands, with interaction as shown below.

SYSGEN cr	Initiate the SYSGEN program.
SYSGEN VERSION m.m	SYSGEN sign-on message.
SOURCE DRIVE NAME (OR RETURN TO SKIP)	Respond with the drive name (one of the letters A, B, C, or D) of the disk containing a CP/M system; usually A. If a copy of CP/M already exists in memory, due to a MOVCPM command, type a cr only. Typing a drive name x will cause the response:
SOURCE ON x THEN TYPE RETURN	Place a diskette containing the CP/M operating system on drive x (x is one of A, B, C, or D). Answer with cr when ready.
FUNCTION COMPLETE	System is copied to memory. SYSGEN will then prompt with:
DESTINATION DRIVE NAME (OR RETURN TO REBOOT)	If a diskette is being initialized, place the new disk into a drive and answer with the drive name. Otherwise, type a cr and the system will reboot from drive A. Typing drive name x will cause SYSGEN to prompt

with:

DESTINATION ON x THEN TYPE RETURN Place new diskette into drive
x; type return when ready.

FUNCTION COMPLETE

New diskette is initialized
in drive x.

The "DESTINATION" prompt will be repeated until a single carriage return is typed at the console, so that more than one disk can be initialized.

Upon completion of a successful system generation, the new diskette contains the operating system, and only the built-in commands are available. A factory-fresh IBM-compatible diskette appears to CP/M as a diskette with an empty directory; therefore, the operator must copy the appropriate COM files from an existing CP/M diskette to the newly constructed diskette using the PIP transient.

The user can copy all files from an existing diskette by typing the PIP command

PIP B: = A: *.*[v] cr

which copies all files from disk drive A to disk drive B, and verifies that each file has been copied correctly. The name of each file is displayed at the console as the copy operation proceeds.

It should be noted that a SYSGEN does not destroy the files which already exist on a diskette; it results only in construction of a new operating system. Further, if a diskette is being used only on drives B through D, and will never be the source of a bootstrap operation on drive A, the SYSGEN need not take place. In fact, a new diskette needs absolutely no initialization to be used with CP/M.

6.7. SUBMIT ufn parm#1 ... parm#n cr

The SUBMIT command allows CP/M commands to be batched together for automatic processing. The ufn given in the SUBMIT command must be the filename of a file which exists on the currently logged disk, with an assumed file type of "SUB." The SUB file contains CP/M prototype commands, with possible parameter substitution. The actual parameters parm#1 ... parm#n are substituted into the prototype commands, and, if no errors occur, the file of substituted commands are processed sequentially by CP/M.

The prototype command file is created using the ED program, with interspersed "\$" parameters of the form

\$1 \$2 \$3 ... \$n

corresponding to the number of actual parameters which will be included when the file is submitted for execution. When the SUBMIT transient is executed, the actual parameters parm#1 ... parm#n are paired with the formal parameters \$1 ... \$n in the prototype commands. If the number of formal and actual parameters does not correspond, then the submit function is aborted with an error message at the console. The SUBMIT function creates a file of substituted commands with the name

\$\$\$SUB

on the logged disk. When the system reboots (at the termination of the SUBMIT), this command file is read by the CCP as a source of input, rather than the console. If the SUBMIT function is performed on any disk other than drive A, the commands are not processed until the disk is inserted into drive A and the system reboots. Further, the user can abort command processing at any time by typing a rubout when the command is read and echoed. In this case, the \$\$\$SUB file is removed, and the subsequent commands come from the console. Command processing is also aborted if the CCP detects an error in any of the commands. Programs which execute under CP/M can abort processing of command files when error conditions occur by simply erasing any existing \$\$\$SUB file.

In order to introduce dollar signs into a SUBMIT file, the user may type a "\$\$" which reduces to a single "\$" within the command file. Further, an up-arrow symbol "↑" may precede an alphabetic character x, which produces a single ctl-x character within the file.

The last command in a SUB file can initiate another SUB file, thus allowing chained batch commands.

Suppose the file ASMBL.SUB exists on disk and contains the prototype commands

```
ASM $1
DIR $1.*
ERA *.BAK
PIP $2:=$1.PRN
ERA $1.PRN
```

and the command

SUBMIT ASMBL X PRN cr

is issued by the operator. The SUBMIT program reads the ASMBL.SUB file, substituting "X" for all occurrences of \$1 and "PRN" for all occurrences of \$2, resulting in a \$\$\$SUB file containing the commands

```

ASM X
DIR X.*
ERA *.BAK
PIP PRN:=X.PRN
ERA X.PRN

```

which are executed in sequence by the CCP.

The SUBMIT function can access a SUB file which is on an alternate drive by preceding the file name by a drive name. Submitted files are only acted upon, however, when they appear on drive A. Thus, it is possible to create a submitted file on drive B which is executed at a later time when it is inserted in drive A.

6.8. DUMP ufn cr

The DUMP program types the contents of the disk file (ufn) at the console in hexadecimal form. The file contents are listed sixteen bytes at a time, with the absolute byte address listed to the left of each line in hexadecimal. Long typeouts can be aborted by pushing the rubout key during printout. (The source listing of the DUMP program is given in the "CP/M Interface Guide" as an example of a program written for the CP/M environment.)

6.9. MOVCPM cr

The MOVCPM program allows the user to reconfigure the CP/M system for any particular memory size. Two optional parameters may be used to indicate (1) the desired size of the new system and (2) the disposition of the new system at program termination. If the first parameter is omitted or a "*" is given, the MOVCPM program will reconfigure the system to its maximum size, based upon the kilobytes of contiguous RAM in the host system (starting at 0000H). If the second parameter is omitted, the system is executed, but not permanently recorded; if "*" is given, the system is left in memory, ready for a SYSGEN operation. The MOVCPM program relocates a memory image of CP/M and places this image in memory in preparation for a system generation operation. The command forms are:

```
MOVCPM cr
```

Relocate and execute CP/M for management of the current memory configuration (memory is examined for contiguous RAM, starting at 100H). Upon completion of the relocation, the new system is executed but not permanently recorded on the diskette. The system which is constructed contains a BIOS for the Intel MDS 800.

MOVCPM n cr	Create a relocated CP/M system for management of an n kilobyte system (n must be in the range 16 to 64), and execute the system, as described above.
MOVCPM * * cr	Construct a relocated memory image for the current memory configuration, but leave the memory image in memory, in preparation for a SYSGEN operation.
MOVCPM n * cr	Construct a relocated memory image for an n kilobyte memory system, and leave the memory image in preparation for a SYSGEN operation.

The command

MOVCPM * *

for example, constructs a new version of the CP/M system and leaves it in memory, ready for a SYSGEN operation. The message

READY FOR "SYSGEN" OR
"SAVE 32 CPMxx.COM"

is printed at the console upon completion, where xx is the current memory size in kilobytes. The operator can then type

SYSGEN cr	Start the system generation.
SOURCE DRIVE NAME (OR RETURN TO SKIP)	Respond with a cr to skip the CP/M read operation since the system is already in memory as a result of the previous MOVCPM operation.
DESTINATION DRIVE NAME (OR RETURN TO REBOOT)	Respond with B to write new system to the diskette in drive B. SYSGEN will prompt with:
DESTINATION ON B, THEN TYPE RETURN	Ready the fresh diskette on drive B and type a return when ready.

Note that if you respond with "A" rather than "B" above, the system will be written to drive A rather than B. SYSGEN will continue to type the prompt:

DESTINATION DRIVE NAME (OR RETURN TO REBOOT)

until the operator responds with a single carriage return, which stops the

SYSGEN program with a system reboot.

The user can then go through the reboot process with the old or new diskette. Instead of performing the SYSGEN operation, the user could have typed

SAVE 32 CPMxx.COM

at the completion of the MOVCPM function, which would place the CP/M memory image on the currently logged disk in a form which can be "patched." This is necessary when operating in a non-standard environment where the BIOS must be altered for a particular peripheral device configuration, as described in the "CP/M System Alteration Guide."

Valid MOVCPM commands are given below:

MOVCPM 48 cr	Construct a 48K version of CP/M and start execution.
MOVCPM 48 * cr	Construct a 48K version of CP/M in preparation for permanent recording; response is READY FOR "SYSGEN" OR "SAVE 32CPM48.COM"
MOVCPM * * cr	Construct a maximum memory version of CP/M and start execution.

It is important to note that the newly created system is serialized with the number attached to the original diskette and is subject to the conditions of the Digital Research Software Licensing Agreement.

7. BDOS ERROR MESSAGES.

There are three error situations which the Basic Disk Operating System intercepts during file processing. When one of these conditions is detected, the BDOS prints the message:

BDOS ERR ON x: error

where x is the drive name, and "error" is one of the three error messages:

BAD SECTOR
SELECT
READ ONLY

The "BAD SECTOR" message indicates that the disk controller electronics has detected an error condition in reading or writing the diskette. This condition is generally due to a malfunctioning disk controller, or an extremely worn diskette. If you find that your system reports this error more than once a month, you should check the state of your controller electronics, and the condition of your media. You may also encounter this condition in reading files generated by a controller produced by a different manufacturer. Even though controllers are claimed to be IBM-compatible, one often finds small differences in recording formats. The MDS-800 controller, for example, requires two bytes of one's following the data CRC byte, which is not required in the IBM format. As a result, diskettes generated by the Intel MDS can be read by almost all other IBM-compatible systems, while disk files generated on other manufacturer's equipment will produce the "BAD SECTOR" message when read by the MDS. In any case, recovery from this condition is accomplished by typing a `ctl-C` to reboot (this is the safest!), or a return, which simply ignores the bad sector in the file operation. Note, however, that typing a return may destroy your diskette integrity if the operation is a directory write, so make sure you have adequate backups in this case.

The "SELECT" error occurs when there is an attempt to address a drive beyond the A through D range. In this case, the value of x in the error message gives the selected drive. The system reboots following any input from the console.

The "READ ONLY" message occurs when there is an attempt to write to a diskette which has been designated as read-only in a `STAT` command, or has been set to read-only by the BDOS. In general, the operator should reboot CP/M either by using the warm start procedure (`ctl-C`) or by performing a cold start whenever the diskettes are changed. If a changed diskette is to be read but not written, BDOS allows the diskette to be changed without the warm or cold start, but internally marks the drive as read-only. The status of the drive is subsequently changed to read/write if a warm or cold start occurs. Upon issuing this message, CP/M waits for input from the console. An automatic warm start takes place following any input.

8. OPERATION OF CP/M ON THE MDS.

This section gives operating procedures for using CP/M on the Intel MDS microcomputer development system. A basic knowledge of the MDS hardware and software systems is assumed.

CP/M is initiated in essentially the same manner as Intel's ISIS operating system. The disk drives are labelled 0 through 3 on the MDS, corresponding to CP/M drives A through D, respectively. The CP/M system diskette is inserted into drive 0, and the BOOT and RESET switches are depressed in sequence. The interrupt 2 light should go on at this point. The space bar is then depressed on the device which is to be taken as the system console, and the light should go out (if it does not, then check connections and baud rates). The BOOT switch is then turned off, and the CP/M signon message should appear at the selected console device, followed by the "A>" system prompt. The user can then issue the various resident and transient commands

The CP/M system can be restarted (warm start) at any time by pushing the INT 0 switch on the front panel. The built-in Intel ROM monitor can be initiated by pushing the INT 7 switch (which generates a RST 7), except when operating under DDT, in which case the DDT program gets control instead.

Diskettes can be removed from the drives at any time, and the system can be shut down during operation without affecting data integrity. Note, however, that the user must not remove a diskette and replace it with another without rebooting the system (cold or warm start), unless the inserted diskette is "read only."

Due to hardware hang-ups or malfunctions, CP/M may type the message

BDOS ERR ON x: BAD SECTOR

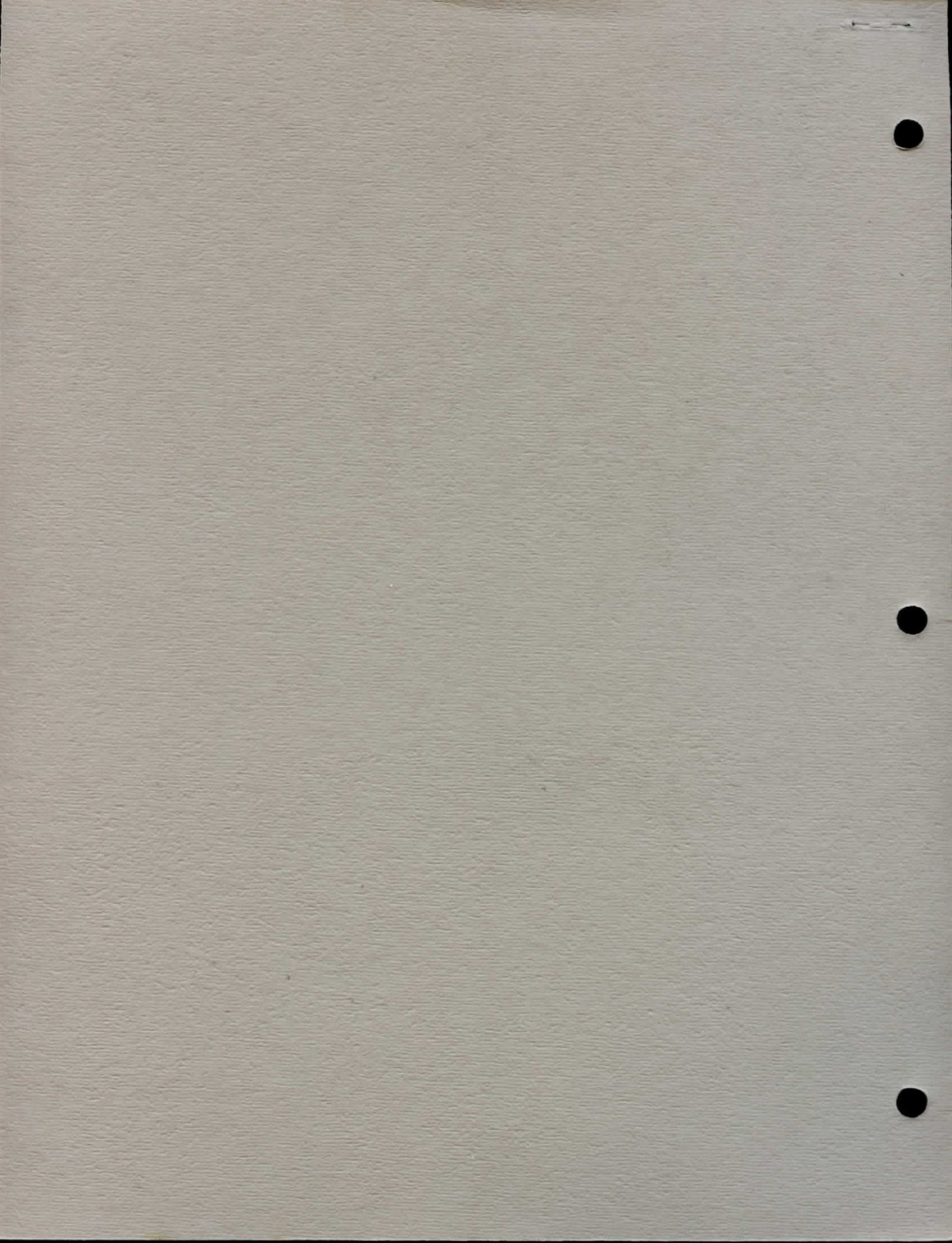
where x is the drive which has a permanent error. This error may occur when drive doors are opened and closed randomly, followed by disk operations, or may be due to a diskette, drive, or controller failure. The user can optionally elect to ignore the error by typing a single return at the console. The error may produce a bad data record, requiring re-initialization of up to 128 bytes of data. The operator can reboot the CP/M system and try the operation again.

Termination of a CP/M session requires no special action, except that it is necessary to remove the diskettes before turning the power off, to avoid random transients which often make their way to the drive electronics.

It should be noted that factory-fresh IBM-compatible diskettes should be used rather than diskettes which have previously been used with any ISIS version. In particular, the ISIS "FORMAT" operation produces non-standard sector numbering throughout the diskette. This non-standard numbering seriously degrades the performance of CP/M, and will operate noticeably slower

than the distribution version. If it becomes necessary to reformat a diskette (which should not be the case for standard diskettes), a program can be written under CP/M which causes the MDS 800 controller to reformat with sequential sector numbering (1-26) on each track.

Note: "MDS 800" and "ISIS" are registered trademarks of Intel Corporation.





Post Office Box 579, Pacific Grove, California 93950, (408) 373-3403

CP/M INTERFACE GUIDE

COPYRIGHT (c) 1976, 1978

DIGITAL RESEARCH

TABLE OF CONTENTS

1.	INTRODUCTION	1
1.1	CP/M Organization	1
1.2	Operation of Transient Programs	1
1.3	Operating System Facilities	3
2.	BASIC I/O FACILITIES	4
2.1	Direct and Buffered I/O	5
2.2	A Simple Example	5
3.	DISK I/O FACILITIES	9
3.1	File System Organization	9
3.2	File Control Block Format	10
3.3	Disk Access Primitives	12
3.4	Random Access	18
4.	SYSTEM GENERATION	18
4.1	Initializing CP/M from an Existing Diskette . . .	19
5.	CP/M ENTRY POINT SUMMARY	20
6.	ADDRESS ASSIGNMENTS	22
7.	SAMPLE PROGRAMS	23

CP/M INTERFACE GUIDE

1. INTRODUCTION

This manual describes the CP/M system organization including the structure of memory, as well as system entry points. The intention here is to provide the necessary information required to write programs which operate under CP/M, and which use the peripheral and disk I/O facilities of the system.

1.1 CP/M Organization

CP/M is logically divided into four parts:

- BIOS - the basic I/O system for serial peripheral control
- BDOS - the basic disk operating system primitives
- CCP - the console command processer
- TPA - the transient program area

The BIOS and BDOS are combined into a single program with a common entry point and referred to as the FDOS. The CCP is a distinct program which uses the FDOS to provide a human-oriented interface to the information which is cataloged on the diskette. The TPA is an area of memory (i.e. the portion which is not used by the FDOS and CCP) where various non-resident operating system commands are executed. User programs also execute in the TPA. The organization of memory in a standard CP/M system is shown in Figure 1.

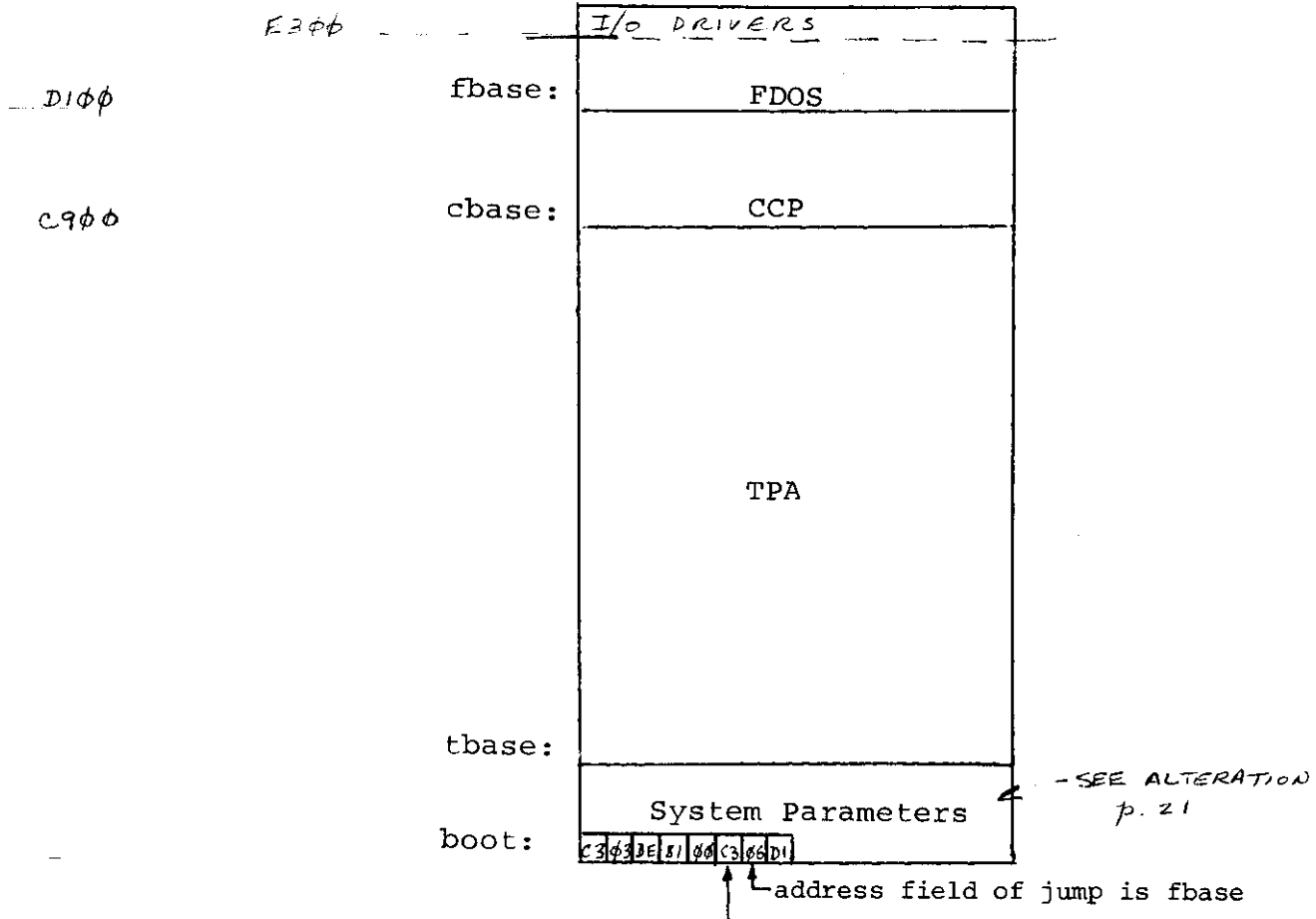
The lower portion of memory is reserved for system information (which is detailed in later sections), including user defined interrupt locations. The portion between tbase and cbase is reserved for the transient operating system commands, while the portion above cbase contains the resident CCP and FDOS. The last three locations of memory contain a jump instruction to the FDOS entry point which provides access to system functions.

1.2 Operation of Transient Programs

Transient programs (system functions and user-defined programs) are loaded into the TPA and executed as follows. The operator communicates with the CCP by typing command lines following each prompt character. Each command line takes one of the forms:

$$\left\{ \begin{array}{l} \text{<command>} \\ \text{<command> <filename>} \\ \text{<command> <filename>.<filetype>} \end{array} \right\}$$

Figure 1. CP/M Memory Organization



entry: the principal entry point to FDOS is at location 0005 which contains a JMP to fbase. The address field at location 0006 can be used to determine the size of available memory, assuming the CCP is being overlayed.

Note: The exact addresses for boot, tbase, cbase, fbase, and entry vary with the CP/M version (see Section 6. for version correspondence).

Where <command> is either a built-in command (e.g., DIR or TYPE), or the name of a transient command or program. If the <command> is a built-in function of CP/M, it is executed immediately; otherwise the CCP searches the currently addressed disk for a file by the name

<command>.COM

If the file is found, it is assumed to be a memory image of a program which executes in the TPA, and thus implicitly originates at tbase in memory (see the CP/M LOAD command). The CCP loads the COM file from the diskette into memory starting at tbase, and extending up to address cbase.

If the <command> is followed by either a <filename> or <filename>.<filetype>, then the CCP prepares a file control-block (FCB) in the system information area of memory. This FCB is in the form required to access the file through the FDOS, and is given in detail in Section 3.2.

The program then executes, perhaps using the I/O facilities of the FDOS. If the program uses no FDOS facilities, then the entire remaining memory area is available for data used by the program. If the FDOS is to remain in memory, then the transient program can use only up to location fbase as data.* In any case, if the CCP area is used by the transient, the entire CP/M system must be reloaded upon the transient's completion. This system reload is accomplished by a direct branch to location "boot" in memory.

The transient uses the CP/M I/O facilities to communicate with the operator's console and peripheral devices, including the floppy disk subsystem. The I/O system is accessed by passing a "function number" and an "information address" to CP/M through the address marked "entry" in Figure 1. In the case of a disk read, for example, the transient program sends the number corresponding to a disk read, along with the address of an FCB, and CP/M performs the operation, returning with either a disk read complete indication or an error number indicating that the disk operation was unsuccessful. The function numbers and error indicators are given in detail in Section 3.3.

1.3 Operating System Facilities

CP/M facilities which are available to transients are divided into two categories: BIOS operations, and BDOS primitives. The BIOS operations are listed first:**

* Address "entry" contains a jump to the lowest address in the FDOS, and thus "entry+1" contains the first FDOS address which cannot be overlaid.

**The device support (exclusive of the disk subsystem) corresponds exactly to Intel's peripheral definition, including I/O port assignment and status byte format (see the Intel manual which discusses the Intellec MDS hardware environment).

```

Read Console Character
Write Console Character
Read Reader Character
Write Punch Character
Write List Device Character
Set I/O Status
Interrogate Device Status
Print Console Buffer
Read Console Buffer
Interrogate Console Status

```

The exact details of BIOS access are given in Section 2. The BDOS primitives include the following operations:

```

Disk System Reset
Drive Select
File Creation
File Open
File Close
Directory Search
File Delete
File Rename
Read Record
Write Record
Interrogate Available Disks
Interrogate Selected Disk
Set DMA Address

```

The details of BDOS access are given in Section 3.

2. BASIC I/O FACILITIES

Access to common peripherals is accomplished by passing a function number and information address to the BIOS. 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 BIOS when a value is returned:

```

DECLARE ENTRY LITERALLY    '0005H'; /* MONITOR ENTRY */

MON2:  PROCEDURE (FUNC, INFO) BYTE;
        DECLARE FUNC BYTE, INFO ADDRESS;
        GO TO ENTRY;

        END MON2;

```

or

```

MON1:  PROCEDURE (FUNC,INFO);
        DECLARE FUNC BYTE, INFO ADDRESS;
        GO TO ENTRY;
        END MON1

```

if no returned value is expected.

2.1 Direct and Buffered I/O.

The BIOS entry points are given in Table I. In the case of simple character I/O to the console, the BIOS reads the console device, and removes the parity bit. The character is echoed back to the console, and tab characters (control-I) are expanded to tab positions starting at column one and separated by eight character positions. The I/O status byte takes the form shown in Table I, and can be programmatically interrogated or changed. The buffered read operation takes advantage of the CP/M line editing facilities. That is, the program sends the address of a read buffer whose first byte is the length of the buffer. The second byte is initially empty, but is filled-in by CP/M to the number of characters read from the console after the operation (not including the terminating carriage-return). The remaining positions are used to hold the characters read from the console. The BIOS line editing functions which are performed during this operation are given below:

```

break      - line delete and transmit
rubout     - delete last character typed, and echo
control-C  - system reboot
control-U  - delete entire line
control-E  - return carriage, but do not transmit
             buffer (physical carriage return)
<cr>      - transmit buffer

```

The read routine also detects control character sequences other than those shown above, and echos them with a preceding "!" symbol. The print entry point allows an entire string of symbols to be printed before returning from the BIOS. The string is terminated by a "\$" symbol.

2.2 A Simple Example

As an example, consider the following PL/M procedures and procedure calls which print a heading, and successively read the console buffer. Each console buffer is then echoed back in reverse order:

```

PRINTCHAR:  PROCEDURE (B);
    /* SEND THE ASCII CHARACTER B TO THE CONSOLE */
    DECLARE B BYTE;
    CALL MON1(2,B);
    END PRINTCHAR;

CRLF:  PROCEDURE;
    /* SEND CARRIAGE-RETURN-LINE-FEED CHARACTERS */
    CALL PRINTCHAR (0DH); CALL PRINTCHAR (0AH);
    END CRLF;

PRINT:  PROCEDURE (A);
    /* PRINT THE BUFFER STARTING AT ADDRESS A */
    DECLARE A ADDRESS;
    CALL MON1(9,A);
    END PRINT;

DECLARE  RDBUFF (130) BYTE;

READ:  PROCEDURE;
    /* READ CONSOLE CHARACTERS INTO 'RDBUFF' */
    RDBUFF=128; /* FIRST BYTE SET TO BUFFER LENGTH */
    CALL MON1(10,.RDBUFF);
    END READ;

DECLARE I BYTE;
CALL CRLF;
CALL PRINT (.'TYPE INPUT LINES $');
DO WHILE 1; /* INFINITE LOOP-UNTIL CONTROL-C */
    CALL CRLF; CALL PRINTCHAR ('*'); /* PROMPT WITH '*' */
    CALL READ; I = RDBUFF(1);
    DO WHILE (I := I -1) <> 255;
        CALL PRINTCHAR (RDBUFF(I+2));
    END;
END;

```

The execution of this program might proceed as follows:

```

TYPE INPUT LINES
*HELLO,
OLLEH
*WALL WALLA WASH,
HSAW ALLAW ALLAW
*MOM WOW,
WOW MOM
*↑C                      (system reboot)

```

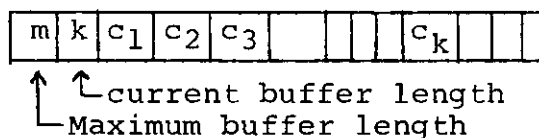
TABLE I
BASIC I/O OPERATIONS

FUNCTION/ NUMBER	ENTRY PARAMETERS	RETURNED VALUE	TYPICAL CALL
Read Console 1 => C=1H	None	ASCII Character =< A = CHR	I = MON2(1,0)
Write Console 2 => C=2H	ASCII Character => E=CHR	None	CALL MON1(2,'A')
Read Reader 3 => C=3H	None	ASCII Character =< A = CHR	I = MON2(3,0)
Write Punch 4 => C=4H	ASCII Character	None	CALL MON1(4,'B')
Write List 5 => C=5H	ASCII Character => E=CHR	None	CALL MON1(5,'C')
Get I/O Status 7 => C=7H	None	I/O Status Byte =< A = I/O BYTE	IOSTAT=MON2(7,0)
Set I/O Status 8 => C=8H	I/O Status Byte => E = I/O BYTE	None	CALL MON1(8,IOSTAT)
Print Buffer 9 => C=9H	Address of string termi- nated by '\$' => DE = STRING ADRS	None	CALL MON1(9, . 'PRINT THIS \$')

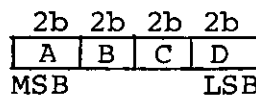
TABLE I (continued)

FUNCTION/ NUMBER	ENTRY PARAMETERS	RETURNED VALUE	TYPICAL CALL
Read Buffer 10 $\Rightarrow C = \phi AH$	Address of Read Buffer* (See Note ₁) $\Rightarrow DE = \text{BUFFER ADRES}$	Read buffer is filled to maxi- mum length with console charac- ters $\Leftarrow (\text{BUFFER FULL})$	CALL MON1(10, .RDBUFF);
Interrogate Console Ready 11 $\Rightarrow C = \phi BH$	None	Byte value with least signifi- cant bit = 1 (true) if con- sole character is ready $\Leftarrow A = \phi FF (\text{IF READY})$ $A = \phi (\text{IF NOT READY})$	I = MON2(11,0)

Note₁: Read buffer is a sequence of memory locations of the form:



Note₂: The I/O status byte is defined as three fields A, B, C, and D



requiring two bits each, listed from most significant to least significant bit, which define the current device assignment as follows:

$$\begin{array}{l}
 \text{D} = \left\{ \begin{array}{l} 0 \text{ TTY} \\ 1 \text{ CRT} \\ 2 \text{ BATCH} \\ 3 \text{ -} \end{array} \right\} \quad
 \text{C} = \left\{ \begin{array}{l} 0 \text{ TTY} \\ 1 \text{ FAST READER} \\ 2 \text{ -} \\ 3 \text{ -} \end{array} \right\} \quad
 \text{B} = \left\{ \begin{array}{l} 0 \text{ TTY} \\ 1 \text{ FAST PUNCH} \\ 2 \text{ -} \\ 3 \text{ -} \end{array} \right\} \quad
 \text{A} = \left\{ \begin{array}{l} 0 \text{ TTY} \\ 1 \text{ CRT} \\ 2 \text{ -} \\ 3 \text{ -} \end{array} \right\}
 \end{array}$$

Console Reader Punch List

3. DISK I/O FACILITIES

The BDOS section of CP/M provides access to files stored on diskettes. The discussion which follows gives the overall file organization, along with file access mechanisms.

3.1 File Organization

CP/M implements a named file structure on each diskette, providing a logical organization which allows any particular file to contain any number of records, from completely empty, to the full capacity of a diskette. Each diskette is logically distinct, with a complete operating system, disk directory, and file data area. The disk file names are in two parts: the <filename> which can be from one to eight alphanumeric characters, and the <filetype> which consists of zero through three alphanumeric characters. The <filetype> names the generic category of a particular file, while the <filename> distinguishes a particular file within the category. The <filetype>s listed below give some generic categories which have been established, although they are generally arbitrary:

ASM	assembler source file
PRN	assembler listing file
HEX	assembler or PL/M machine code in "hex" format
BAS	BASIC Source file
INT	BASIC Intermediate file
COM	Memory image file (i.e., "Command" file for transients, produced by LOAD)
BAK	Backup file produced by editor (see ED manual)
\$\$\$	Temporary files created and normally erased by editor and utilities

Thus, the name

X.ASM

is interpreted as an assembly language source file by the CCP with <filename> X.

The files in CP/M are organized as a logically contiguous sequence of 128 byte records (although the records may not be physically contiguous on the diskette), which are normally read or written in sequential order. Random access is allowed under CP/M however, as described in Section 3.4. No particular format within records is assumed by CP/M, although some transients expect particular formats:

- (1) Source files are considered a sequence of ASCII characters, where each "line" of the source file is followed by carriage-return-line-feed characters. Thus, one 128 byte CP/M record could contain several logical lines of source text. Machine code "hex" tapes are also assumed to be in this format, although the loader does not require the carriage-return-line-feed characters. End of text is given by the character control-z, or real end-of-file returned by CP/M.

and

- (2) COM files are assumed to be absolute machine code in memory image form, starting at tbase in memory. In this case, control-z is not considered an end of file, but instead is determined by the actual space allocated to the file being accessed.

3.2 File Control Block Format

Each file being accessed through CP/M has a corresponding file control block (FCB) which provides name and allocation information for all file operations. The FCB is a 33-byte area in the transient program's memory space which is set up for each file. The FCB format is given in Figure 2. When accessing CP/M files, it is the programmer's responsibility to fill the lower 16 bytes of the FCB, along with the CR field. Normally, the FN and FT fields are set to the ASCII <filename> and <filetype>, while all other fields are set to zero. Each FCB describes up to 16K bytes of a particular file (0 to 128 records of 128 bytes each), and, using automatic mechanisms of CP/M, up to 15 additional extensions of the file can be addressed. Thus, each FCB can potentially describe files up to 256K bytes (which is slightly larger than the diskette capacity).

FCB's are stored in a directory area of the diskette, and are brought into central memory before file operations (see the OPEN and MAKE commands) then updated in memory as file operations proceed, and finally recorded on the diskette at the termination of the file operation (see the CLOSE command). This organization makes CP/M file organization highly reliable, since diskette file integrity can only be disrupted in the unlikely case of hardware failure during update of a single directory entry.

It should be noted that the CCP constructs an FCB for all transients by scanning the remainder of the line following the transient name for a <filename> or <filename>.<filetype> combination. Any field not specified is assumed to be all blanks. A properly formed FCB is set up at location tfcb (see Section 6), with an assumed I/O buffer at tbuff. The transient can use tfcb as an address in subsequent input or output operations on this file.

In addition to the default fcb which is set-up at address tfcb, the CCP also constructs a second default fcb at address tfcb+16 (i.e., the disk map field of the fcb at tbase). Thus, if the user types

PROGNAME X.ZOT Y.ZAP

the file PROGNAME.COM is loaded to the TPA, and the default fcb at tfcb is initialized to the filename X with filetype ZOT. Since the user typed a second file name, the 16 byte area beginning at tfcb + 16₁₀ is also initialized with the filename Y and filetype ZAP. It is the responsibility of the program to move this second filename and filetype to another area (usually a separate file control block) before opening the file which begins at tbase, since the open operation will fill the disk map portion, thus overwriting the second name and type.

If no file names were specified in the original command, then the fields beginning at tfcb and tfcb + 16 both contain blanks (20H). If one file name was specified, then the field at tfcb + 16 contains blanks. If the filetype is omitted, then the field is assumed to contain blanks. In all cases, the CCP translates lower case alphabetic to upper case to be consistent with the CP/M file naming conventions.

As an added programming convenience, the default buffer at tbuff is initialized to hold the entire command line past the program name. Address tbuff contains the number of characters, and tbuff+1, tbuff+2, ..., contain the remaining characters up to, but not including, the carriage return. Given that the above command has been typed at the console, the area beginning at tbuff is set up as follows:

tbuff:

+0	+1	+2	+3	+4	+5	+6	+7	+8	+9	+10	+11	+12	+13	+14	+15
12	Ø	X	.	Z	O	T	Ø	Y	.	Z	A	P	?	?	?

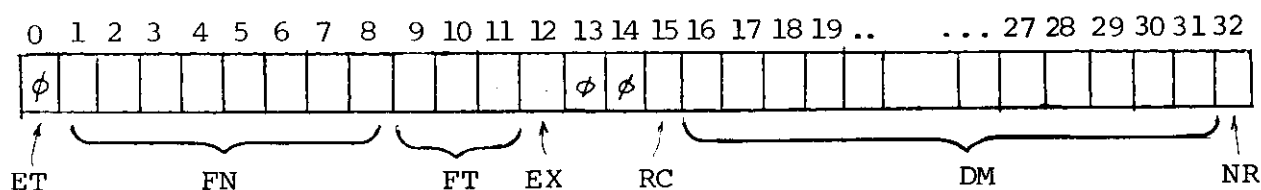
where 12 is the number of valid characters (in binary), and Ø represents an ASCII blank. Characters are given in ASCII upper case, with uninitialized memory following the last valid character.

Again, it is the responsibility of the program to extract the information from this buffer before any file operations are performed since the FDOS uses the tbuff area to perform directory functions.

In a standard CP/M system, the following values are assumed:

boot:	0000H	bootstrap load (warm start)
entry:	0005H	entry point to FDOS
tfcb:	005CH	first default file control block
tfcb+16	006CH	second file name
tbuff	0080H	default buffer address
tbase:	0100H	base of transient area

Figure 2. File Control Block Format



<u>FIELD</u>	<u>FCB POSITIONS</u>	<u>PURPOSE</u>
ET	0	Entry type (currently not used, but assumed zero)
FN	1-8	File name, padded with ASCII blanks
FT	9-11	File type, padded with ASCII blanks
EX	12	File extent, normally set to zero
	13-14	Not used, but assumed zero
RC	15	Record count is current extent Size (0 to 128 records)
DM	16-31	Disk allocation map, filled-in and used by CP/M
NR	32	Next record number to read, or write

3.3 Disk Access Primitives

Given that a program has properly initialized the FCB's for each of its files, there are several operations which can be performed, as shown in Table II. In each case, the operation is applied to the currently selected disk (see the disk select operation in Table II), using the file information in a specific FCB. The following PL/M program segment, for example, copies the contents of the file X.Y to the (new) file NEW.FIL:

```

DECLARE RET BYTE;

OPEN:      PROCEDURE (A)
            DECLARE A ADDRESS;
            RET=MON2(15,A);
            END OPEN;

CLOSE:     PROCEDURE (A);
            DECLARE A ADDRESS;
            RET=MON2(16,A);
            END;

MAKE:      PROCEDURE (A);
            DECLARE A ADDRESS;
            RET=MON2(22,A);
            END MAKE;

DELETE:    PROCEDURE (A);
            DECLARE A ADDRESS;
            /* IGNORE RETURNED VALUE */
            CALL MON1(19,A);
            END DELETE;

READBF:    PROCEDURE (A);
            DECLARE A ADDRESS;
            RET=MON2(20,A);
            END READBF;

WRITEBF:   PROCEDURE (A);
            DECLARE A ADDRESS;
            RET=MON2(21,A);
            END WRITEBF;

INIT:      PROCEDURE;
            CALL MON1(13,0);
            END INIT;

/* SET UP FILE CONTROL BLOCKS */
DECLARE FCB1 (33) BYTE
            INITIAL (0,'X      ','Y  ',0,0,0,0),
            FCB2 (33) BYTE
            INITIAL (0,'NEW    ','FIL',0,0,0,0);

```

```

CALL INIT;
/* ERASE 'NEW.FIL' IF IT EXISTS */
CALL DELETE (.FCB2);
/* CREATE 'NEW.FIL' AND CHECK SUCCESS */
CALL MAKE (.FCB2);
IF RET = 255 THEN CALL PRINT (.'NO DIRECTORY SPACE $');
ELSE
DO; /* FILE SUCCESSFULLY CREATED, NOW OPEN 'X.Y' */
CALL OPEN (.FCB1);
IF RET = 255 THEN CALL PRINT (.'FILE NOT PRESENT $');
ELSE
DO; /* FILE X.Y FOUND AND OPENED, SET
NEXT RECORD TO ZERO FOR BOTH FILES */
FCB1(32), FCB2(32) = 0;
/* READ FILE X.Y UNTIL EOF OR ERROR */
CALL READBF (.FCB1); /*READ TO 80H*/
DO WHILE RET = 0;
CALL WRITEBF (.FCB2) /*WRITE FROM 80H*/
IF RET = 0 THEN /*GET ANOTHER RECORD*/
CALL READBF (.FCB1); ELSE
CALL PRINT (.'DISK WRITE ERROR $');
END;
IF RET < >1 THEN CALL PRINT (.'TRANSFER ERROR $');
ELSE
DO; CALL CLOSE (.FCB2);
IF RET = 255 THEN CALL PRINT (.'CLOSE ERROR$');
END;
END;
END;
EOF

```

This program consists of a number of utility procedures for opening, closing, creating, and deleting files, as well as two procedures for reading and writing data. These utility procedures are followed by two FCB's for the input and output files. In both cases, the first 16 bytes are initialized to the <filename> and <filetype> of the input and output files. The main program first initializes the disk system, then deletes any existing copy of "NEW.FIL" before starting. The next step is to create a new directory entry (and empty file) for "NEW.FIL". If file creation is successful, the input file "X.Y" is opened. If this second operation is also successful, then the disk to disk copy can proceed. The NR fields are set to zero so that the first record of each file is accessed on subsequent disk I/O operations. The first call to READBF fills the (implied) DMA buffer at 80H with the first record from X.Y. The loop which follows copies the record at 80H to "NEW.FIL" and then reports any errors, or reads another 128 bytes from X.Y. This transfer operation continues until either all data has been transferred, or an error condition arises. If an error occurs, it is reported; otherwise the new file is closed and the program halts.

TABLE II
DISK ACCESS PRIMITIVES

FUNCTION/NUMBER	ENTRY PARAMETERS	RETURNED VALUE	TYPICAL CALL
Lift Head 12 => C = 0CH	None	None Head is lifted from current drive	CALL MON2(12,0)
Initialize BDOS and select disk "A" Set DMA address to 80H 13 => C = 0DH	None	None Side effect is that disk A is "logged- in" while all others are considered "off- line"	CALL MON1(13,0)
Log-in and select disk X 14 => C = 0EH	An integer value cor- responding to the disk to log-in: A=0, B=1, C=2, etc. => E = DISK UNIT	None Disk X is considered "on-line" and selec- ted for subsequent file operations	CALL MON1(14,1) (log-in disk "B")
Open file 15 => C = 0FH	Address of the FCB for the file to be accessed => DE = FCB ADRES	Byte address of the FCB in the directory, if found, or 255 if file not present. The DM bytes are set by the BDOS. =< A = 0FFH IF NOT FOUND	I = MON2(15,.FCB)
Close file 16 => C = 10H	Address of an FCB which has been pre- viously created or opened => DE = FCB ADRES	Byte address of the directory entry cor- responding to the FCB, or 255 if not present =< A = 0FFH IF ERROR	I = MON2(16,.FCB)

TABLE II (continued)

FUNCTION/NUMBER	ENTRY PARAMETERS	RETURNED VALUE	TYPICAL CALL
Search for file 17	Address of FCB containing <filename> and <filetype> to match. ASCII "?" in FCB matches any character. => DE = FCB ADRES	Byte address of first FCB in directory that matches input FCB, if any; otherwise 255 indicates no match. $\Leftarrow A = \emptyset FFFH$ IF NOT FOUND	I = MON2(17,.FCB)
Search for next occurrence 18	Same as above, but called after function 17 (no other intermediate BDOS calls allowed) => DE = FCB ADRES	Byte address of next $\Leftarrow A = (SAME AS 17)$	I = MON2(18,.FCB)
Delete File 19	Address of FCB containing <filename> and <filetype> of file to delete from diskette => DE = FCB ADRES	None	I = MON2(19,.FCB)
Read Next Record 20	Address of FCB of a successfully OPENED file, with NR set to the next record to read (see note ₁) => DE = FCB ADRES	0 = successful read 1 = read past end of file 2 = reading unwritten data in random access $\Leftarrow A = (ABOVE)$	I = MON2(20,.FCB)

Note₁: The I/O operations transfer data to/from address 80H for the next 128 bytes unless the DMA address has been altered (see function 26). Further, the NR field of the FCB is automatically incremented after the operation. If the NR field exceeds 128, the next extent is opened automatically, and the NR field is reset to zero.

TABLE II (continued)

FUNCTION/NUMBER	ENTRY PARAMETERS	RETURNED VALUE	TYPICAL CALL
Write Next Record 21	Same as above, except NR is set to the next record to write	0 = successful write 1 = error in extending file 2 = end of disk data 255 = no more directory space (see note ₂)	I = MON2(21,.FCB)
=> C = 15H	=> DE = FCB ADRES	<= A = (ABOVE)	
Make File 22	Address of FCB with <filename> and <file-type> set. Directory entry is created, the file is initialized to empty.	Byte address of directory entry allocated to the FCB, or 255 if no directory space is available	I = MON2(22,.FCB)
=> C = 16H	=> DE = FCB ADRES	<= A = 0FFH IF NO SPACE IN DIR	
Rename FCB 23	Address of FCB with old FN and FT in first 16 bytes, and new FN and FT in second 16 bytes	Address of the directory entry which matches the first 16 bytes. The <filename> and <file-type> is altered 255 if no match.	I = MON2(23,.FCB)
=> C = 17H	=> DE = FCB ADRES SETUP AS ABOVE	<= A = 0FFH IF NOT FOUND	

Note₂: There are normally 64 directory entries available on each diskette (can be expanded to 255 entries), where one entry is required for the primary file, and one for each additional extent.

TABLE II (continued)

FUNCTION/NUMBER	ENTRY PARAMETERS	RETURNED VALUE	TYPICAL CALL
Interrogate log- in vector 24	None	Byte value with "1" in bit positions of "on line" disks, with least signi- ficant bit corres- ponding to disk "A" ← A = (A60V6)	I = MON2(24,0)
⇒ C = 18H			
Set DMA address 26	Address of 128 byte DMA buffer	None Subsequent disk I/O takes place at spe- cified address in memory	CALL MON1(26,2000H)
⇒ C = 1AH	⇒ DE = BUFFER ADDRESS		
Interrogate Allocation 27	None	Address of the allo- cation vector for the current disk (used by STATUS com- mand)	MON3: PROCEDURE(...) ADDRESS; A = MON3(27,0);
⇒ C = 1BH		← HL = ALLOC VECTOR	
Interrogate Drive number 25	None	Disk number of currently logged disk (i.e., the drive which will be used for the next disk operation ← A = UNIT NUMBER	I = MON2(25,0);
⇒ C = 19H			

TYPE B:NOTES.VAN

TITLE SECRETS OF CP/M VERSION 1.4
FILENAME SECRET14.DOC
AUTHOR Robert A. Van Valzah 4/27/78

M Y S T E R Y B I O S F U N C T I O N S

FUNC	ACTION
28	MAKE CURRENT DRIVE READ ONLY (R/O)
29	RETURN R/O VECTOR. RETURNS A BYTE VALUE (IN REG A) WITH A "1" IN BIT POSITIONS OF R/O DISKS, LEAST SIGNIFICANT BIT CORRESPONDING TO DRIVE A.
30	SET DIRECTORY I/O DMA ADDRESS. INITIALLY SET TO 80H. DISK DATA I/O CONTINUES AT THE LAST ADDRESS SET BY FUNCTION 26.

B O O T A N D G O M O D F O R C C P

There are some dedicated CP/M applications where it would be nice if a cold start would load CP/M and then load a transient and give it control. This could be used to make a disk which would always power up in BASIC; for instance. This capability has been built into CP/M version 1.4.

THEORY

When the CCP is loaded, it checks its console input buffer to see if a command is waiting. As supplied by Digital Research, this buffer is empty and the CCP simply signs on and prompts the user for input. The task at hand, therefore, is to fill this buffer so its contents will be executed as a CCP command as the CCP is loaded. Assuming you have SID and desire a command length of <=16 characters, this is quite simple:

- 1) Load an image of your operating system into low memory (ready for SYSGEN). EX: SID CPMnn.COM
- 2) Type "I" followed by the command line as you like CCP to execute it. EX: ISTAT.COM
- 3) Type M30,90,987
- 4) Boot and SYSGEN a disk in the normal manner.

The disk will now boot and execute the command typed in step 2 above.

If you don't have SID or desire a command line longer than 16 characters:

- 1) Perform step 1 above (use DDT if needed).
- 2) Patch the length of your command into location 987H.
- 3) Patch the ASCII of the command into memory starting at

location 988H, following it with the Digital Research
copyright message "COPYRIGHT (C) 1978, DIGITAL
RESEARCH".

4) Perform step 4 above.

3.4 Random Access

Recall that a single FCB describes up to a 16K segment of a (possibly) larger file. Random access within the first 16K segment is accomplished by setting the NR field to the record number of the record to be accessed before the disk I/O takes place. Note, however, that if the 128th record is written, then the BDOS automatically increments the extent field (EX), and opens the next extent, if possible. In this case, the program must explicitly decrement the EX field and re-open the previous extent. If random access outside the first 16K segment is necessary, then the extent number e be explicitly computed, given an absolute record number r as

$$e = \left\lfloor \frac{r}{128} \right\rfloor$$

or equivalently,

$$e = \text{SHR}(r, 7)$$

this extent number is then placed in the EX field before the segment is opened. The NR value n is then computed as

$$n = r \bmod 128$$

or

$$n = r \text{ AND } 7\text{FH}.$$

When the programmer expects considerable cross-segment accesses, it may save time to create an FCB for each of the 16K segments, open all segments for access, and compute the relevant FCB from the absolute record number r .

4. SYSTEM GENERATION

As mentioned previously, every diskette used under CP/M is assumed to contain the entire system (excluding transient commands) on the first two tracks. The operating system need not be present, however, if the diskette is only used as secondary disk storage on drives B, C, ..., since the CP/M system is loaded only from drive A.

The CP/M file system is organized so that an IBM-compatible diskette from the factory (or from a vendor which claims IBM compatibility) looks like a diskette with an empty directory. Thus, the user must first copy a version of the CP/M system from an existing diskette to the first two tracks of the new diskette, followed by a sequence of copy operations, using PIP, which transfer the transient command files from the original diskette to the new diskette.

NOTE: before you begin the CP/M copy operation, read your Licensing Agreement. It gives your exact legal obligations when making reproductions of CP/M in whole or in part, and specifically requires that you place the copyright notice

Copyright (c), 1976
Digital Research

on each diskette which results from the copy operation.

4.1. Initializing CP/M from an Existing Diskette

The first two tracks are placed on a new diskette by running the transient command SYSGEN, as described in the document "An Introduction to CP/M Features and Facilities." The SYSGEN operation brings the CP/M system from an initialized diskette into memory, and then takes the memory image and places it on the new diskette.

Upon completion of the SYSGEN operation, place the original diskette on drive A, and the initialized diskette on drive B. Reboot the system; the response should be

A>

indicating that drive A is active. Log into drive B by typing

B:

and CP/M should respond with

B>

indicating that drive B is active. If the diskette in drive B is factory fresh, it will contain an empty directory. Non-standard diskettes may, however, appear as full directories to CP/M, which can be emptied by typing

ERA *.*

when the diskette to be initialized is active. Do not give the ERA command if you wish to preserve files on the new diskette since all files will be erased with this command.

After examining disk B, reboot the CP/M system and return to drive A for further operations.

The transient commands are then copied from drive A to drive B using the PIP program. The sequence of commands shown below, for example, copy the principal programs from a standard CP/M diskette to the new diskette:

```
A>PIP,
*B:STAT.COM=STAT.COM,
*B:PIP.COM=PIP.COM,
*B:LOAD.COM=LOAD.COM,
*B:ED.COM=ED.COM,
```

```

*B:ASM.COM=ASM.COM,
*B:SYSGEN.COM=SYSGEN.COM,
*B:DDT.COM=DDT.COM,
*,
A>

```

The user should then log in disk B, and type the command

```
DIR *.*,
```

to ensure that the files were transferred to drive B from drive A. The various programs can then be tested on drive B to check that they were transferred properly.

Note that the copy operation can be simplified somewhat by creating a "submit" file which contains the copy commands. The file could be named GEN.SUB, for example, and might contain

```

SYSGEN,
PIP B:STAT.COM=STAT.COM,
PIP B:PIP.COM=PIP.COM,
PIP B:LOAD.COM=LOAD.COM,
PIP B:ED.COM=ED.COM,
PIP B:ASM.COM=ASM.COM,
PIP B:SYSGEN.COM=SYSGEN.COM,
PIP B:DDT.COM=DDT.COM,

```

The generation of a new diskette from the standard diskette is then done by typing simply

```
SUBMIT GEN,
```

5. CP/M ENTRY POINT SUMMARY

The functions shown below summarize the functions of the FDOS. The function number is passed in Register C (first parameter in PL/M), and the information is passed in Registers D,E (second PL/M parameter). Single byte results are returned in Register A. If a double byte result is returned, then the high-order byte comes back in Register B (normal PL/M return). The transient program enters the FDOS through location "entry" (see Section 7.) as shown in Section 2. for PL/M, or

CALL entry

in assembly language. All registers are altered in the FDOS.

<u>Function</u>	<u>Number</u>	<u>Information</u>	<u>Result</u>
0	System Reset		
1	Read Console		ASCII character
2	Write Console	ASCII character	
3	Read Reader		ASCII character
4	Write Punch	ASCII character	
5	Write List	ASCII character	
6	(not used)		
7	Interrogate I/O Status		I/O Status Byte
8	Alter I/O Status	I/O Status Byte	
9	Print Console Buffer	Buffer Address	
10	Read Console Buffer	Buffer Address	
11	Check Console Status		True if character Ready
12	Lift Disk Head		
13	Reset Disk System		
14	Select Disk	Disk number	
15	Open File	FCB Address	Completion Code
16	Close File	" "	" "
17	Search First	" "	" "
18	Search Next	" "	" "
19	Delete File	" "	" "
20	Read Record	" "	" "
21	Write Record	" "	" "
22	Create File	" "	" "
23	Rename File	" "	" "
24	Interrogate Login		Login Vector
25	Interrogate Disk		Selected Disk Number
26	Set DMA Address	DMA Address	
27	Interrogate Allocation		Address of Allocation Vector

6. ADDRESS ASSIGNMENTS

The standard distribution version of CP/M is organized for an Intel MDS microcomputer developmental system with 16K of main memory, and two diskette drives. Larger systems are available in 16K increments, providing management of 32K, 48K, and 64K systems (the largest MDS system is 62K since the ROM monitor provided with the MDS resides in the top 2K of the memory space). For each additional 16K increment, add 4000H to the values of cbase and fbase.

The address assignments are

boot = 0000H	warm start operation
tfcb = 005CH	default file control block location
tbuff= 0080H	default buffer location
tbase= 0100H	base of transient program area
cbase= 2900H	base of console command processor
fbase= 3200H	base of disk operating system
entry= 0005H	entry point to disk system from user programs

7. SAMPLE PROGRAMS

This section contains two sample programs which interface with the CP/M operating system. The first program is written in assembly language, and is the source program for the DUMP utility. The second program is the CP/M LOAD utility, written in PL/M.

The assembly language program begins with a number of "equates" for system entry points and program constants. The equate

```
BDOS EQU 0005H
```

for example, gives the CP/M entry point for peripheral I/O functions. The default file control block address is also defined (FCB), along with the default buffer address (BUFF). Note that the program is set up to run at location 100H, which is the base of the transient program area. The stack is first set-up by saving the entry stack pointer into OLDSP, and resetting SP to the local stack. The stack pointer upon entry belongs to the console command processor, and need not be saved unless control is to return to the CCP upon exit. That is, if the program terminates with a reboot (branch to location 0000H) then the entry stack pointer need not be saved.

The program then jumps to MAIN, past a number of subroutines which are listed below:

- BREAK - when called, checks to see if there is a console character ready. BREAK is used to stop the listing at the console
- PCHAR - print the character which is in register A at the console.
- CRLF - send carriage return and line feed to the console
- PNIB - print the hexadecimal value in register A in ASCII at the console
- PHEX - print the byte value (two ASCII characters) in register A at the console
- ERR - print error flag #n at the console, where n is
 - 1 if file cannot be opened
 - 2 if disk read error occurred
- GNB - get next byte of data from the input file. If the IBP (input buffer pointer) exceeds the size of the input buffer, then another disk record of 128 bytes is read. Otherwise, the next character in the buffer is returned. IBP is updated to point to the next character

The MAIN program then appears, which begins by calling SETUP. The SETUP subroutine, discussed below, opens the input file and checks for errors. If the file is opened properly, the GLOOP (get loop) label gets control.

On each successive pass through the GLOOP label, the next data byte is fetched using GNB and save in register B. The line addresses are listed every sixteen bytes, so there must be a check to see if the least significant 4 bits is zero on each output. If so, the line address is taken from registers h and l, and typed at the left of the line. In all cases, the byte which was previously saved in register B is brought back to register A, following label NONUM, and printed in the output line. The cycle through GLOOP continues until an end of file condition is detected in DISKR, as described below. Thus, the output lines appear as

```
0000  bb bb bb bb bb bb bb bb bb bb bb bb bb bb bb
0010  bb bb bb bb bb bb bb bb bb bb bb bb bb bb bb
...
```

until the end of file.

The label FINIS gets control upon end of file. CRLF is called first to return the carriage from the last line output. The CCP stack pointer is then reclaimed from OLDSP, followed by a RET to return to the console command processor. Note that a JMP 0000H could be used following the FINIS label, which would cause the CP/M system to be brought in again from the diskette (this operation is necessary only if the CCP has been overlaid by data areas).

The file control block format is then listed (FCBDN ... FCBLN) which overlays the fcb at location 05CH which is setup by the CCP when the DUMP program is initiated. That is, if the user types

DUMP X.Y

then the CCP sets up a properly formed fcb at location 05CH for the DUMP (or any other) program when it goes into execution. Thus, the SETUP subroutine simply addresses this default fcb, and calls the disk system to open it. The DISKR (disk read) routine is called whenever GNB needs another buffer full of data. The default buffer at location 80H is used, along with a pointer (IBP) which counts bytes as they are processed. Normally, an end of file condition is taken as either an ASCII LAH (control-z), or an end of file detection by the DOS. The file dump program, however, stops only on a DOS end of file.

```

; FILE DUMP PROGRAM, READS AN INPUT FILE AND PRINTS IN HEX
;
; COPYRIGHT (C), DIGITAL RESEARCH, 1975, 1976
;

```

25

```

0100      ORG      100H
0005 =    BDOS    EQU      0005H      ;DOS ENTRY POINT
000F =    OPENF   EQU      15        ;FILE OPEN
0014 =    READF   EQU      20        ;READ FUNCTION
0002 =    TYPEF   EQU      2         ;TYPE FUNCTION
0001 =    CONS    EQU      1         ;READ CONSOLE
000B =    BRKF    EQU      11        ;BREAK KEY FUNCTION (TRUE IF CHAR READY)
;
005C =    FCB     EQU      5CH        ;FILE CONTROL BLOCK ADDRESS
0080 =    BUFF    EQU      80H        ;INPUT DISK BUFFER ADDRESS
;
; SET UP STACK
0100 210000 LXI      H,0
0103 39      DAD     SP
0104 220F01  SHLD    OLDSP
0107 315101  LXI     SP,STKTOP
010A C3C401  JMP     MAIN
;
; VARIABLES
010D      IBP:    DS      2           ;INPUT BUFFER POINTER
;
; STACK AREA
010F      OLDSP:  DS      2
0111      STACK: DS      64
0151 =    STKTOP  EQU     $
;
; SUBROUTINES
;
BREAK: ;CHECK BREAK KEY (ACTUALLY ANY KEY WILL DO)
      PUSH H! PUSH D! PUSH B; ENVIRONMENT SAVED
      MVI     C,BRKF
      CALL    BDOS
      POP B! POP D! POP H; ENVIRONMENT RESTORED
      RET
;
PCHAR: ;PRINT A CHARACTER
      PUSH H! PUSH D! PUSH B; SAVED
      MVI     C,TYPEF
      MOV     E,A
      CALL    BDOS
      POP B! POP D! POP H; RESTORED
      RET
;
CRLF:
016A 3E0D    MVI     A,0DH
016C CD5D01  CALL    PCHAR
016F 3E0A    MVI     A,0AH
0171 CD5D01  CALL    PCHAR
0174 C9      RET
;
;
PNIB: ;PRINT NIBBLE IN REG A
0175 E60F    ANI     0FH          ;LOW 4 BITS
0177 FE0A    CPI     10
0179 D28101  JNC     P10

```

```

; LESS THAN OR EQUAL TO 9
017C C630 ADI '0'
017E C38301 JMP PRN

;
; GREATER OR EQUAL TO 10
0181 C637 P10: ADI 'A' - 10
0183 CD5D01 PRN: CALL PCHAR
0186 C9 RET

;
PHEX: ;PRINT HEX CHAR IN REG A
0187 F5 PUSH PSW
0188 0F RRC
0189 0F RRC
018A 0F RRC
018B 0F RRC
018C CD7501 CALL PNIB ;PRINT NIBBLE
018F F1 POP PSW
0190 CD7501 CALL PNIB
0193 C9 RET

;
ERR: ;PRINT ERROR MESSAGE
0194 CD6A01 CALL CRLF
0197 3E23 MVI A, '#'
0199 CD5D01 CALL PCHAR
019C 78 MOV A,B
019D C630 ADI '0'
019F CD5D01 CALL PCHAR
01A2 CD6A01 CALL CRLF
01A5 C3F701 JMP FINIS

;
GNB: ;GET NEXT BYTE
01A8 3A0D01 LDA IBP
01AB FE80 CPI 80H
01AD C2B401 JNZ G0

; READ ANOTHER BUFFER
;
;
01B0 CD1602 CALL DISKR
01B3 AF XRA A

;
G0: ;READ THE BYTE AT BUFF+REG A
01B4 5F MOV E,A
01B5 1600 MVI D,0
01B7 3C INR A
01B8 320D01 STA IBP

; POINTER IS INCREMENTED
; SAVE THE CURRENT FILE ADDRESS
01BB E5 PUSH H
01BC 218000 LXI H,BUFF
01BF 19 DAD D
01C0 7E MOV A,M

; BYTE IS IN THE ACCUMULATOR
;
; RESTORE FILE ADDRESS AND INCREMENT
01C1 E1 POP H
01C2 23 INX H
01C3 C9 RET

;
MAIN: ; READ AND PRINT SUCCESSIVE BUFFERS
01C4 CDFF01 CALL SETUP ;SET UP INPUT FILE

```

```

01C7 3E80          MVI      A,80H
01C9 320D01        STA      IBP      ;SET BUFFER POINTER TO 80H
01CC 21FFFF        LXI      H,0FFFFH ;SET TO -1 TO START

;
GLOOP:
01CF CDA801        CALL     GNB
01D2 47            MOV      B,A
; PRINT HEX VALUES

; CHECK FOR LINE FOLD
01D3 7D            MOV      A,L
01D4 E60F          ANI      0FH      ;CHECK LOW 4 BITS
01D6 C2EB01        JNZ      NONUM
; PRINT LINE NUMBER
01D9 CD6A01        CALL     CRLF
;
; CHECK FOR BREAK KEY
01DC CD5101        CALL     BREAK
01DF 0F            RRC
01E0 DAF701        JC       FINIS    ;DON'T PRINT ANY MORE
;
01E3 7C            MOV      A,H
01E4 CD8701        CALL     PHEX
01E7 7D            MOV      A,L
01E8 CD8701        CALL     PHEX
NONUM:
01EB 3E20          MVI      A,' '
01ED CD5D01        CALL     PCHAR
01F0 78            MOV      A,B
01F1 CD8701        CALL     PHEX
01F4 C3CF01        JMP      GLOOP
;
EPSA: ;END PSA
; END OF INPUT
FINIS:
01F7 CD6A01        CALL     CRLF
01FA 2A0F01        LHLD     OLDSP
01FD F9            SPHL
01FE C9            RET
;
;
; FILE CONTROL BLOCK DEFINITIONS
005C = FCB DN EQU FCB+0 ;DISK NAME
005D = FCB FN EQU FCB+1 ;FILE NAME
0065 = FCB FT EQU FCB+9 ;DISK FILE TYPE (3 CHARACTERS)
0068 = FCB RL EQU FCB+12 ;FILE'S CURRENT REEL NUMBER
006B = FCB RC EQU FCB+15 ;FILE'S RECORD COUNT (0 TO 128)
007C = FCB CR EQU FCB+32 ;CURRENT (NEXT) RECORD NUMBER (0 TO 127)
007D = FCB LN EQU FCB+33 ;FCB LENGTH
;
;
SETUP: ;SET UP FILE
; OPEN THE FILE FOR INPUT
01FF 115C00        LXI      D,FCB
0202 0E0F          MVI      C,OPENF
0204 CD0500        CALL     BDOS
; CHECK FOR ERRORS
0207 FEFF          CPI      255
0209 C21102        JNZ      OPNOK

```

```

020C 0601      ;      BAD OPEN
020E CD9401    MVI      B,1      ;OPEN ERROR
               CALL      ERR
;
OPNOK: ;OPEN IS OK.
0211 AF        XRA      A
0212 327C00    STA      FCBCR
0215 C9        RET
;
DISKR: ;READ DISK FILE RECORD
0216 E5D5C5    PUSH H! PUSH D! PUSH B
0219 115C00    LXI      D,FCB
021C 0E14      MVI      C,READF
021E CD0500    CALL     BDOS
0221 C1D1E1    POP B! POP D! POP H
0224 FE00      CPI      0      ;CHECK FOR ERRS
0226 C8        RZ
;
0227 FE01      MAY BE EOF
0229 CAF701    CPI      1
               JZ       FINIS
;
022C 0602      MVI      B,2      ;DISK READ ERROR
022E CD9401    CALL     ERR
;
0231          END

```

The PL/M program which follows implements the CP/M LOAD utility. The function is as follows. The user types

LOAD filename,

If filename.HEX exists on the diskette, then the LOAD utility reads the "hex" formatted machine code file and produces the file

filename.COM

where the COM file contains an absolute memory image of the machine code, ready for load and execution in the TPA. If the file does not appear on the diskette, the LOAD program types

SOURCE IS READER

and reads an Addmaster paper tape reader which contains the hex file.

The LOAD program is set up to load and run in the TPA, and, upon completion, return to the CCP without rebooting the system. Thus, the program is constructed as a single procedure called LOADCOM which takes the form

```
OFAH:
  LOADCOM: PROCEDURE;
    /* LIBRARY PROCEDURES */
    MON1: ...
    /* END LIBRARY PROCEDURES */
    MOVE: ...
    GETCHAR: ...
    PRINTNIB: ...
    PRINTHEX: ...
    PRINTADDR: ...
    RELOC: ...
    {
      SETMEM:
      READHEX:
      READBYTE:
      READCS:
      MAKEDOUBLE:
      DIAGNOSE:
    }
    END RELOC;

    DECLARE STACK(16) ADDRESS, SP ADDRESS;
    SP = STACKPTR; STACKPTR = .STACK(LENGTH(STACK));

    ...
    CALL RELOC;
    ...
    STACKPTR = SP;
    RETURN 0;
  END LOADCOM;
;
EOF
```

The label OFAH at the beginning sets the origin of the compilation to OFAH, which causes the first 6 bytes of the compilation to be ignored when loaded (i.e., the TPA starts at location 100H and thus OFAH,...,OFFH are deleted from the COM file). In a PL/M compilation, these 6 bytes are used to set up the stack pointer and branch around the subroutines in the program. In this case, there is only one subroutine, called LOADCOM, which results in the following machine memory image for LOAD

```

OFAH: LXI SP,plmstack      ;SET SP TO DEFAULT STACK
OFDH: JMP pastsubr         ;JUMP AROUND LOADCOM
100H: beginning of LOADCOM procedure
      ....
      end of LOADCOM procedure
      RET

pastsubr:
      EI
      HLT

```

Since the machine code between OFAH and OFFH is deleted in the load, execution actually begins at the top of LOADCOM. Note, however, that the initialization of the SP to the default stack has also been deleted; thus, there is a declaration and initialization of an explicit stack and stack pointer before the call to RELOC at the end of LOADCOM. This is necessary only if we wish to return to the CCP without a reboot operation: otherwise the origin of the program is set to 100H, the declaration of LOADCOM as a procedure is not necessary, and termination is accomplished by simply executing a

```
GO TO 0000H;
```

at the end of the program. Note also that the overhead for a system reboot is not great (approximately 2 seconds), but can be bothersome for system utilities which are used quite often, and do not need the extra space.

The procedures listed in LOADCOM as "library procedures" are a standard set of PL/M subroutines which are useful for CP/M interface. The RELOC procedure contains several nested subroutines for local functions, and actually performs the load operation when called from LOADCOM. Control initially starts on line 327 where the stackpointer is saved and re-initialized to the local stack. The default file control block name is copied to another file control block (SFCB) since two files may be open at the same time. The program then calls SEARCH to see if the HEX file exists; if not, then the high speed reader is used. If the file does exist, it is opened for input (if possible). The filetype COM is moved to the default file control block area, and any existing copies of filename.COM files are removed from the diskette before creating a new file. The MAKE operation creates a new file, and, if successful, RELOC is called to read the HEX file and produce the COM file. At the end of processing by RELOC, the COM file is closed (line 350). Note that the HEX file does not need to be closed since it was opened for input only. The data written to a file is not permanently recorded until the file is successfully closed.

Disk input characters are read through the procedure GETCHAR on line 137. Although the DMA facilities of CP/M could be used here, the GETCHAR procedure instead uses the default buffer at location 80H and moves each buffer into a vector called SBUFF (source buffer) as it is read. On exit, the GETCHAR procedure returns the next input character and updates the source buffer pointer (SBP).

The SETMEM procedure on line 191 performs the opposite function from GETCHAR. The SETMEM procedure maintains a buffer of loaded machine code in pure binary form which acts as a "window" on the loaded code. If there is an attempt by RELOC to write below this window, then the data is ignored. If the data is within the window, then it is placed into MBUFF (memory buffer). If the data is to be placed above this window, then the window is moved up to the point where it would include the data address by writing the memory image successively (by 128 byte buffers), and moving the base address of the window. Using this technique, the programmer can recover from checksum errors on the high-speed reader by stopping the reader, rewinding the tape for some distance, then restarting LOAD (in this case, LOADING is resumed by interrupting with a NOP instruction). Again, the SETMEM procedure uses the default buffer at location 80H to perform the disk output by moving 128 byte segments to 80H through 0FFH before each write.

```

00001 1
00002 1 0FAH: DECLARE BDOS LITERALLY '0005H';
00003 1 /* TRANSIENT COMMAND LOADER PROGRAM
00004 1
00005 1 COPYRIGHT (C) DIGITAL RESEARCH
00006 1 JUNE, 1975
00007 1 */
00008 1
00009 1 LOADCOM: PROCEDURE BYTE;
00010 2 DECLARE FCBA ADDRESS INITIAL(5CH);
00011 2 DECLARE FCB BASED FCBA (33) BYTE;
00012 2
00013 2 DECLARE BUFFA ADDRESS INITIAL(80H), /* I/O BUFFER ADDR
ESS */
00014 2 BUFFER BASED BUFFA (128) BYTE;
00015 2
00016 2 DECLARE SFCB(33) BYTE, /* SOURCE FILE CONTROL BLOCK */
/
00017 2 BSIZE LITERALLY '1024',
00018 2 EOFILE LITERALLY '1AH',
00019 2 SBUFF(BSIZE) BYTE /* SOURCE FILE BUFFER */
00020 2 INITIAL(EOFILE),
00021 2 RFLAG BYTE, /* READER FLAG */
00022 2 SBP ADDRESS; /* SOURCE FILE BUFFER POINTER
*/
00023 2
00024 2 /* LOADCOM LOADS TRANSIENT COMMAND FILES TO THE DISK F
ROM THE
00025 2 CURRENTLY DEFINED READER PERIPHERAL. THE LOADER PLACE
S THE MACH
00026 2 CODE INTO A FILE WHICH APPEARS IN THE LOADCOM COMMAND
*/
00027 2 /* ***** LIBRARY PROCEDURES FOR DISKIO *****
***** */
00028 2
00029 2 MON1: PROCEDURE(F,A);
00030 3 DECLARE F BYTE,
00031 3 A ADDRESS;
00032 3 GO TO BDOS;
00033 3 END MON1;
00034 2
00035 2 MON2: PROCEDURE(F,A) BYTE;
00036 3 DECLARE F BYTE,
00037 3 A ADDRESS;
00038 3 GO TO BDOS;
00039 3 END MON2;
00040 2
00041 2 READRDR: PROCEDURE BYTE;
00042 3 /* READ CURRENT READER DEVICE */
00043 3 RETURN MON2(3,0);
00044 3 END READRDR;
00045 2
00046 2 DECLAKE
00047 2 TRUE LITERALLY '1',
00048 2 FALSE LITERALLY '0',
00049 2 FOREVER LITERALLY 'WHILE TRUE',
00050 2 CR LITERALLY '13',

```

```
00051 2      LF LITERALLY '10',
00052 2      WHAT LITERALLY '63';
00053 2
00054 2      PRINTCHAR: PROCEDURE(CHAR);
00055 3          DECLARE CHAR BYTE;
00056 3          CALL MON1(2,CHAR);
00057 3          END PRINTCHAR;
00058 2
00059 2      CRLF: PROCEDURE;
00060 3          CALL PRINTCHAR(CR);
00061 3          CALL PRINTCHAR(LF);
00062 3          END CRLF;
00063 2
00064 2      PRINT: PROCEDURE(A);
00065 3          DECLARE A ADDRESS;
00066 3          /* PRINT THE STRING STARTING AT ADDRESS A UNTIL THE
00067 3          NEXT DOLLAR SIGN IS ENCOUNTERED */
00068 3          CALL CRLF;
00069 3          CALL MON1(9,A);
00070 3          END PRINT;
00071 2
00072 2      DECLARE DCNT BYTE;
00073 2
00074 2      INITIALIZE: PROCEDURE;
00075 3          CALL MON1(13,0);
00076 3          END INITIALIZE;
00077 2
00078 2      SELECT: PROCEDURE(D);
00079 3          DECLARE D BYTE;
00080 3          CALL MON1(14,D);
00081 3          END SELECT;
00082 2
00083 2      OPEN: PROCEDURE(FCB);
00084 3          DECLARE FCB ADDRESS;
00085 3          DCNT = MON2(15,FCB);
00086 3          END OPEN;
00087 2
00088 2      CLOSE: PROCEDURE(FCB);
00089 3          DECLARE FCB ADDRESS;
00090 3          DCNT = MON2(16,FCB);
00091 3          END CLOSE;
00092 2
00093 2      SEARCH: PROCEDURE(FCB);
00094 3          DECLARE FCB ADDRESS;
00095 3          DCNT = MON2(17,FCB);
00096 3          END SEARCH;
00097 2
00098 2      SEARCHN: PROCEDURE;
00099 3          DCNT = MON2(18,0);
00100 3          END SEARCHN;
00101 2
00102 2      DELETE: PROCEDURE(FCB);
00103 3          DECLARE FCB ADDRESS;
00104 3          CALL MON1(19,FCB);
00105 3          END DELETE;
00106 2
00107 2      DISKREAD: PROCEDURE(FCB) BYTE;
00108 3          DECLARE FCB ADDRESS;
00109 3          RETURN MON2(20,FCB);
00110 3          END DISKREAD;
```

```

00111 2
00112 2 DISKWRITE: PROCEDURE(FCB) BYTE;
00113 3 DECLARE FCB ADDRESS;
00114 3 RETURN MON2(21,FCB);
00115 3 END DISKWRITE;
00116 2
00117 2 MAKE: PROCEDURE(FCB);
00118 3 DECLARE FCB ADDRESS;
00119 3 DCNT = MON2(22,FCB);
00120 3 END MAKE;
00121 2
00122 2 RENAME: PROCEDURE(FCB);
00123 3 DECLARE FCB ADDRESS;
00124 3 CALL MON1(23,FCB);
00125 3 END RENAME;
00126 2
00127 2 /* ***** END OF LIBRARY PROCEDURES ***** */
***** */
00128 2
00129 2 MOVE: PROCEDURE(S,D,N);
00130 3 DECLARE (S,D) ADDRESS, N BYTE,
00131 3 A BASED S BYTE, B BASED D BYTE;
00132 3 DO WHILE (N:=N-1) <> 255;
00133 3 B = A; S=S+1; D=D+1;
00134 4 END;
00135 3 END MOVE;
00136 2
00137 2 GETCHAR: PROCEDURE BYTE;
00138 3 /* GET NEXT CHARACTER */
00139 3 DECLARE I BYTE;
00140 3 IF RFLAG THEN RETURN READRDR;
00141 3 IF (SBP := SBP+1) <= LAST(SBUFF) THEN
00142 3 RETURN SBUFF(SBP);
00143 3 /* OTHERWISE READ ANOTHER BUFFER FULL */
00144 3 DO SBP = 0 TO LAST(SBUFF) BY 128;
00145 3 IF (I:=DISKREAD(.SFCB)) = 0 THEN
00146 4 CALL MOVE(80H,.SBUFF(SBP),80H); ELSE
00147 4 DO; IF I<>1 THEN CALL PRINT(.DISK READ ER
ROR$');
00148 5 SBUFF(SBP) = EOFIL;
00149 5 SBP = LAST(SBUFF);
00150 5 END;
00151 4 END;
00152 3 SBP = 0; RETURN SBUFF;
00153 3 END GETCHAR;
00154 2 DECLARE
00155 2 STACKPOINTER LITERALLY 'STACKPTR';
00156 2
00157 2
00158 2 PRINTNIB: PROCEDURE(N);
00159 3 DECLARE N BYTE;
00160 3 IF N > 9 THEN CALL PRINTCHAR(N+'A'-10); ELSE
00161 3 CALL PRINTCHAR(N+'0');
00162 3 END PRINTNIB;
00163 2
00164 2 PRINTHEX: PROCEDURE(B);
00165 3 DECLARE B BYTE;
00166 3 CALL PRINTNIB(SHR(B,4)); CALL PRINTNIB(B AND 0FH);
00167 3 END PRINTHEX;
00168 2

```

```

00169 2 PRINTADDR: PROCEDURE(A);
00170 3     DECLARE A ADDRESS;
00171 3     CALL PRINTEX(HIGH(A)); CALL PRINTEX(LOW(A));
00172 3     END PRINTADDR;
00173 2
00174 2
00175 2     /* INTEL HEX FORMAT LOADER */
00176 2
00177 2 RELOC: PROCEDURE;
00178 3     DECLARE (RL, CS, RT) BYTE;
00179 3     DECLARE
00180 3         LA ADDRESS,      /* LOAD ADDRESS */
00181 3         TA ADDRESS,      /* TEMP ADDRESS */
00182 3         SA ADDRESS,      /* START ADDRESS */
00183 3         FA ADDRESS,      /* FINAL ADDRESS */
00184 3         NB ADDRESS,      /* NUMBER OF BYTES LOADED */
00185 3         SP ADDRESS,      /* STACK POINTER UPON ENTRY TO REL
OC */
00186 3
00187 3     MBUFF(256) BYTE,
00188 3     P BYTE,
00189 3     L ADDRESS;
00190 3
00191 3     SETMEM: PROCEDURE(B);
00192 4         /* SET MBUFF TO B AT LOCATION LA MOD LENGTH(MBUFF)
*/
00193 4         DECLARE (B,I) BYTE;
00194 4         IF LA < L THEN /* MAY BE A RETRY */ RETURN;
00195 4         DO WHILE LA > L + LAST(MBUFF); /* WRITE A PARA
GRAPH */
00196 4             DO I = 0 TO 127; /* COPY INTO BUFFER */
00197 5                 BUFFER(I) = MBUFF(LOW(L)); L = L + 1;
00198 6                 END;
00199 5             /* WRITE BUFFER ONTO DISK */
00200 5             P = P + 1;
00201 5             IF DISKWRITE(FCBA) <> 0 THEN
00202 5                 DO; CALL PRINT(. 'DISK WRITE ERRORS');
00203 6                 HALT;
00204 6                 /* RETRY AFTER INTERRUPT NOP */
00205 6                 L = L - 128;
00206 6                 END;
00207 5             END;
00208 4             MBUFF(LOW(LA)) = B;
00209 4             END SETMEM;
00210 3
00211 3     READHEX: PROCEDURE BYTE;
00212 4         /* READ ONE HEX CHARACTER FROM THE INPUT */
00213 4         DECLARE H BYTE;
00214 4         IF (H := GETCHAR) - '0' <= 9 THEN RETURN H - '0';
00215 4         IF H - 'A' > 5 THEN GO TO CHARERR;
00216 4         RETURN H - 'A' + 10;
00217 4         END READHEX;
00218 3
00219 3     READBYTE: PROCEDURE BYTE;
00220 4         /* READ TWO HEX DIGITS */
00221 4         RETURN SHL(READHEX,4) OR READHEX;
00222 4         END READBYTE;
00223 3
00224 3     READCS: PROCEDURE BYTE;
00225 4         /* READ BYTE WHILE COMPUTING CHECKSUM */

```

```

00226 4      DECLARE B BYTE;
00227 4      CS = CS + (B := READBYTE);
00228 4      RETURN B;
00229 4      END READCS;
00230 3
00231 3      MAKE$DOUBLE: PROCEDURE(H,L) ADDRESS;
00232 4      /* CREATE A BOUBLE BYTE VALUE FROM TWO SINGLE BYTE
S */
00233 4      DECLARE (H,L) BYTE;
00234 4      RETURN SHL(DOUBLE(H),8) OR L;
00235 4      END MAKE$DOUBLE;
00236 3
00237 3      DIAGNOSE: PROCEDURE;
00238 4
00239 4      DECLARE M BASED TA BYTE;
00240 4
00241 4      NEWLINE: PROCEDURE;
00242 5      CALL CRLF; CALL PRINTADDR(TA); CALL PRINTCHAR(':')
;
00243 5      CALL PRINTCHAR(' ');
00244 5      END NEWLINE;
00245 4
00246 4      /* PRINT DIAGNOSTIC INFORMATION AT THE CONSOLE */
00247 4      CALL PRINT('LOAD ADDRESS $'); CALL PRINTADDR(TA);
00248 4      CALL PRINT('ERROR ADDRESS $'); CALL PRINTADDR(LA);
00249 4
00250 4      CALL PRINT('BYTES READ:$'); CALL NEWLINE;
00251 4      DO WHILE TA < LA;
00252 4      IF (LOW(TA) AND 0FH) = 0 THEN CALL NEWLINE;
00253 5      CALL PRINTEX(MBUFF(TA-L)); TA=TA+1;
00254 5      CALL PRINTCHAR(' ');
00255 5      END;
00256 4      CALL CRLF;
00257 4      HALT;
00258 4      END DIAGNOSE;
00259 3
00260 3
00261 3      /* INITIALIZE */
00262 3      SA, FA, NB = 0;
00263 3      SP = STACKPOINTER;
00264 3      P = 0; /* PARAGRAPH COUNT */
00265 3      TA,LA,L = 100H; /* BASE ADDRESS OF TRANSIENT ROUTINES
*/
00266 3      IF FALSE THEN
00267 3      CHARERR: /* ARRIVE HERE IF NON-HEX DIGIT IS ENCOU
NTERED */
00268 3      DO; /* RESTORE STACKPOINTER */ STACKPOINTER = SP;
00269 4      CALL PRINT('NON-HEXADECIMAL DIGIT ENCOUNTERED $')
;
00270 4      CALL DIAGNOSE;
00271 4      END;
00272 3
00273 3
00274 3      /* READ RECORDS UNTIL :00XXXX IS ENCOUNTERED */
00275 3
00276 3      DO FOREVER;
00277 3      /* SCAN THE : */
00278 3      DO WHILE GETCHAR <> ':';
00279 4      END;

```

```

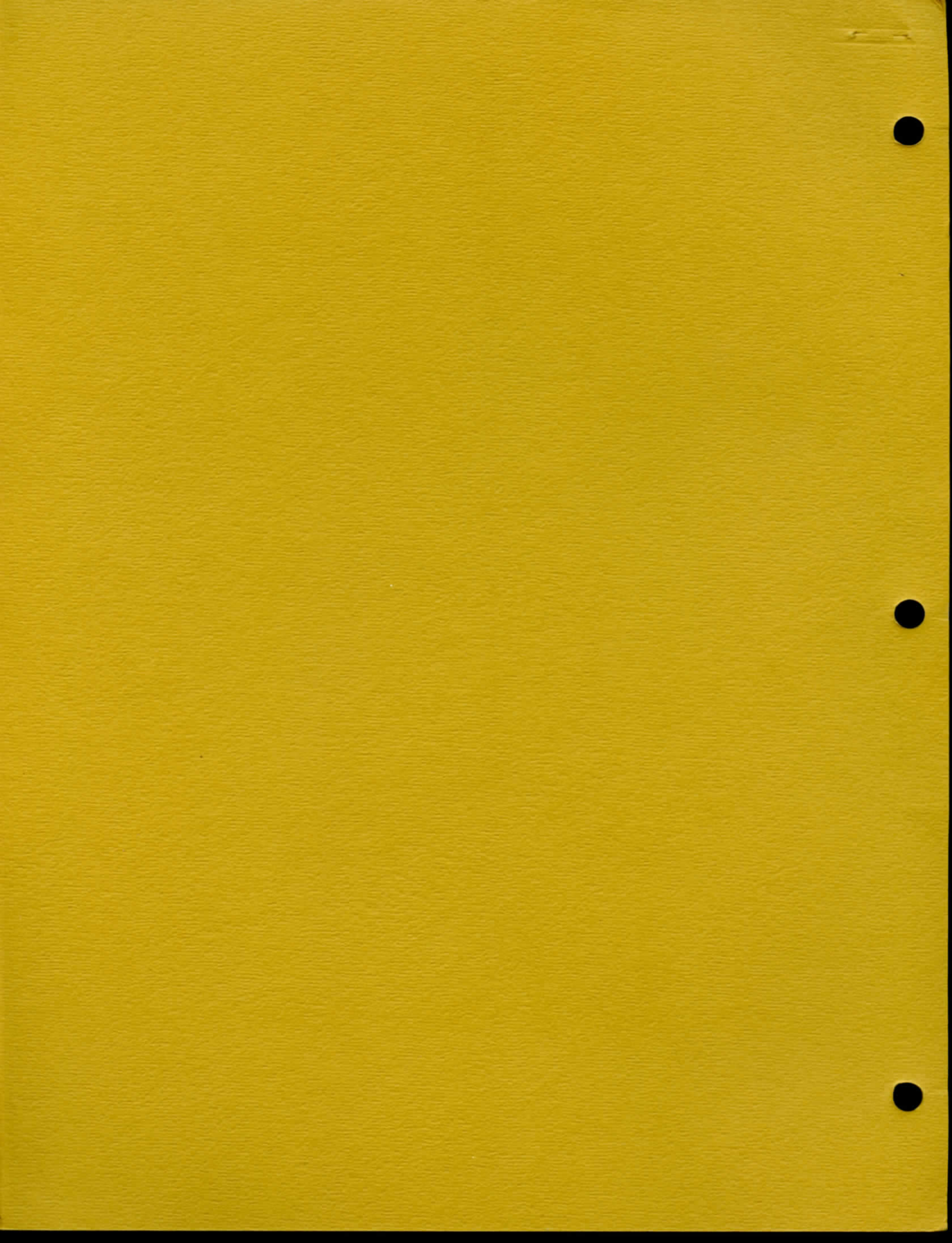
00280 4
00281 4      /* SET CHECK SUM TO ZERO, AND SAVE THE RECORD LENG
TH */
00282 4      CS = 0;
00283 4      /* MAY BE THE END OF TAPE */
00284 4      IF (RL := READCS) = 0 THEN
00285 4          GO TO FIN;
00286 4      NB = NB + RL;
00287 4
00288 4      TA, LA = MAKE$DOUBLE(READCS,READCS);
00289 4      IF SA = 0 THEN SA = LA;
00290 4
00291 4
00292 4      /* READ THE RECORD TYPE (NOT CURRENTLY USED) */
00293 4      RT = READCS;
00294 4
00295 4      /* PROCESS EACH BYTE */
00296 4          DO WHILE (RL := RL - 1) <> 255;
00297 4              CALL SETMEM(READCS); LA = LA+1;
00298 5              END;
00299 4      IF LA > FA THEN FA = LA - 1;
00300 4
00301 4      /* NOW READ CHECKSUM AND COMPARE */
00302 4      IF CS + READBYTE <> 0 THEN
00303 4          DO; CALL PRINT(. 'CHECK SUM ERROR S');
00304 5          CALL DIAGNOSE;
00305 5          END;
00306 4      END;
00307 3
00308 3      FIN:
00309 3      /* EMPTY THE BUFFERS */
00310 3      TA = LA;
00311 3          DO WHILE L < TA;
00312 3              CALL SETMEM(0); LA = LA+1;
00313 4              END;
00314 3      /* PRINT FINAL STATISTICS */
00315 3      CALL PRINT(. 'FIRST ADDRESS S'); CALL PRINTADDR(SA);
00316 3      CALL PRINT(. 'LAST ADDRESS S'); CALL PRINTADDR(FA);
00317 3      CALL PRINT(. 'BYTES READ S'); CALL PRINTADDR(NB);
00318 3      CALL PRINT(. 'RECORDS WRITTEN S'); CALL PRINTHEX(P);
00319 3      CALL CRLF;
00320 3
00321 3      END FLOC;
00322 2
00323 2      /* ARRIVE HERE FROM THE SYSTEM MONITOR, READY TO READ THE~
HEX TAPE
00324 2
00325 2      /* SET UP STACKPOINTER IN THE LOCAL AREA */
00326 2      DECLARE STACK(16) ADDRESS, SP ADDRESS;
00327 2      SP = STACKPOINTER; STACKPOINTER = .STACK(LENGTH(STACK));
00328 2
00329 2      SBB = LENGTH(SBOFF);
00330 2      /* SET UP THE SOURCE FILE */
00331 2      CALL MOVE(FCBA,.SFCB,33);
00332 2      CALL MOVE(.('HEX',0),.SFCB(9),4);
00333 2      CALL SEARCH(.SFCB);
00334 2      IF (FFLAG := COUNT = 255) THEN
00335 2          CALL PRINT(. 'SOURCE IS READERS'); ELSE
00336 2          DO; CALL PRINT(. 'SOURCE IS DISK');

```

```

00337 3      CALL OPEN(.SFCB);
00338 3      IF DCNT = 255 THEN CALL PRINT(.'-CANNOT OPEN SOURC
E$');
00339 3      END;
00340 2      CALL CRLF;
00341 2
00342 2      CALL MOVE(.'COM',FCBA+9,3);
00343 2
00344 2      /* REMOVE ANY EXISTING FILE BY THIS NAME */
00345 2      CALL DELETE(FCBA);
00346 2      /* THEN OPEN A NEW FILE */
00347 2      CALL MAKE(FCBA); FCB(32) = 0; /* CREATE AND SET NEXT RECORD */
00348 2      IF DCNT = 255 THEN CALL PRINT(.'NO MORE DIRECTORY SPACES'
); ELSE
00349 2      DO; CALL RELOC;
00350 3      CALL CLOSE(FCBA);
00351 3      IF DCNT = 255 THEN CALL PRINT(.'CANNOT CLOSE FILES
');
00352 3      END;
00353 2      CALL CRLF;
00354 2
00355 2      /* RESTORE STACKPOINTER FOR RETURN */
00356 2      STACKPOINTER = SP;
00357 2      RETURN 0;
00358 2      END LOADCOM;
00359 1      ;
00360 1      EOF

```



DIGITAL RESEARCH

Post Office Box 579, Pacific Grove, California 93950, (408) 649-3896

CP/M SYSTEM ALTERATION GUIDE

COPYRIGHT (c) 1976, 1978

DIGITAL RESEARCH

REVISION OF JANUARY 1978

Table of Contents

Section	Page
1. INTRODUCTION	1
2. FIRST LEVEL SYSTEM REGENERATION	2
3. SECOND LEVEL SYSTEM REGENERATION	6
4. SAMPLE GETSYS AND PUTSYS PROGRAMS	10
5. DISKETTE ORGANIZATION	12
6. THE BIOS ENTRY POINTS	14
7. A SAMPLE BIOS	20
8. A SAMPLE COLD START LOADER	20
9. RESERVED LOCATIONS IN PAGE ZERO	21
10.. NOTES FOR USERS OF CP/M VERSION 1.3	23
Appendix	
A. THE MDS LOADER MOVE PROGRAM	
B. THE MDS COLD START LOADER	
C. THE MDS BASIC I/O SYSTEM (BIOS)	
D. A SKELETAL CBIOS	
E. A SKELETAL GETSYS/PUTSYS PROGRAM	
F. A SKELETAL COLD START LOADER	

CP/M System Alteration Guide

1. INTRODUCTION

The standard CP/M system assumes operation on an Intel MDS microcomputer development system, but is designed so that the user can alter a specific set of subroutines which define the hardware operating environment. In this way, the user can produce a diskette which operates with a non-standard (but IBM-compatible format) drive controller and/or peripheral devices.

In order to achieve device independence, CP/M is separated into three distinct modules:

- BIOS - Basic I/O System which is environment dependent
- BDOS - Basic Disk Operating System which is not dependent upon the hardware configuration
- CCP - the Console Command Processor which uses the BDOS

Of these modules, only the BIOS is dependent upon the particular hardware. That is, the user can "patch" the distribution version of CP/M to provide a new BIOS which provides a customized interface between the remaining CP/M modules and the user's own hardware system. The purpose of this document is to provide a step-by-step procedure for patching the new BIOS into CP/M.

The new BIOS requires some relatively simple software development and testing; the current BIOS, however, is listed in Appendix C, and can be used as a model for the customized package. A skeletal version of the BIOS is given in Appendix D which can form the base for a modified BIOS. In addition to the BIOS, the user must write a simple memory loader, called GETSYS, which brings the operating system into memory. In order to patch the new BIOS into CP/M, the user must write the reverse of GETSYS, called PUTSYS, which places an altered version of CP/M back onto the diskette. PUTSYS is usually derived from GETSYS by changing the disk read commands into disk write commands. Sample skeletal GETSYS and PUTSYS programs are described in Section 3, and listed in Appendix E. In order to make the CP/M system work automatically, the user must also supply a cold start loader, similar to the one provided with CP/M (listed in Appendices A and B). A skeletal form of a cold start loader is given in Appendix F which can serve as a model for your loader.

2. FIRST LEVEL SYSTEM REGENERATION

The procedure to follow to patch the CP/M system is given below in several steps. Address references in each step are followed by an "H" to denote the hexadecimal radix, and are given for a 16K CP/M system. For larger CP/M systems, add a "bias" to each address which is shown with a "+b" following it, where b is equal to the memory size minus 16K. Values for b in various standard memory sizes are

24K:	b = 24K - 16K = 8K = 02000H
32K:	b = 32K - 16K = 16K = 04000H
40K:	b = 40K - 16K = 24K = 06000H
48K:	b = 48K - 16K = 32K = 08000H
56K:	b = 56K - 16K = 40K = 0A000H
62K:	b = 62K - 16K = 46K = 0B800H
64K:	b = 64K - 16K = 48K = 0C000H

Note: The standard distribution version of CP/M is configured as a 16K system. Therefore, you must first bring up the 16K CP/M system, and then configure it for your actual memory size (see Second Level System Generation).

(1) Review Section 4 and write a GETSYS program which reads the first two tracks of a diskette into memory. The data from the diskette must begin at location 2880H. Code GETSYS so that it starts at location 100H (base of the TPA), as shown in the first part of Appendix E.

(2) Test the GETSYS program by reading a blank diskette into memory, and check to see that the data has been read properly, and that the diskette has not been altered in any way by the GETSYS program.

(3) Run the GETSYS program using an initialized CP/M diskette to see if GETSYS loads CP/M starting at 2880H (the operating system actually starts 128 bytes later at 2900H).

(4) Review Section 4 and write the PUTSYS program which writes memory starting at 2880H back onto the first two tracks of the diskette. The PUTSYS program should be located at 200H, as shown in the second part of Appendix E.

(5) Test the PUTSYS program using a blank uninitialized diskette by writing a portion of memory to the first two tracks; clear memory and read it back using GETSYS. Test PUTSYS completely, since this program will be used to alter CP/M on disk.

(6) Study Sections 5, 6, and 7, along with the distribution version of the BIOS given in Appendix C, and write a simple version which performs a similar function for the customized environment. Use the program given in Appendix D as a model. Call this new BIOS by the name CBIOS (customized BIOS). Implement only the primitive disk operations on a single drive, and

simple console input/output functions in this phase.

(7) Test CBIOS completely to ensure that it properly performs console character I/O and disk reads and writes. Be especially careful to ensure that no disk write operations occur accidentally during read operations, and check that the proper track and sectors are addressed on all reads and writes. Failure to make these checks may cause destruction of the initialized CP/M system after it is patched.

(8) Referring to Figure 1 in Section 5, note that the BIOS is located between locations 3E00H and 3FFFH. Read the CP/M system using GETSYS, and replace the BIOS segment by the new CBIOS developed in step (6) and tested in step (7). This replacement is done in the memory of the machine and will be placed on the diskette in the next step.

(9) Use PUTSYS to place the patched memory image of CP/M onto the first two tracks of a blank diskette for testing.

(10) Use GETSYS to bring the copied memory image from the test diskette back into memory at 2880H, and check to ensure that it has loaded back properly (clear memory, if possible, before the load). Upon successful load, branch to the cold start code at location 3E00H. The cold start routine will initialize page zero, then jump to the CCP (location 2900H) which will call the BDOS, which will call the CBIOS. The CBIOS will be asked to read several sectors on track 2 twice in succession, and, if successful, CP/M will type "A>".

When you make it this far, you are almost on the air. If you have trouble, use whatever debug facilities you have available to trace and breakpoint your CBIOS.

(11) Upon completion of step (10), CP/M has prompted the console for a command input. Test the disk write operation by typing

SAVE 1 X.COM

(recall that all commands must be followed by a carriage return). CP/M should respond with another prompt (after several disk accesses):

A>

If it does not, debug your disk write functions and try again.

(12) Test the directory command by typing

DIR

CP/M should respond with

A: X COM

- (13) Test the erase command by typing

ERA X.COM

CP/M should respond with the A prompt. When you make it this far, you should have an operational system which will only require a bootstrap loader to function completely.

- (14) Write a bootstrap loader which is similar to GETSYS, and place it on track 0, sector 1 using PUTSYS (again using the test diskette, not the distribution diskette). See Sections 5 and 8 for more information on the bootstrap operation.

- (15) Retest the new test diskette with the bootstrap loader installed by executing steps (11), (12), and (13). Upon completion of these tests, type a control-C (control and C keys simultaneously). The system should then execute a "warm start" which reboots the system and types the A> prompt.

- (16) At this point, you probably have a good version of your customized CP/M system on your test diskette. Use GETSYS to load CP/M from your test diskette. Remove the test diskette, place the distribution diskette (or a legal copy) into the drive, and use PUTSYS to replace the distribution version by your customized version. Do not make this replacement if you are unsure of your patch since this step destroys the system which was sent to you from Digital Research.

- (17) Load your modified CP/M system, and test it by typing

DIR

CP/M should respond with a list of files which are provided on the initialized diskette. One such file should be the memory image for the debugger, called DDT.COM.

NOTE: from now on, it is important that you always reboot the CP/M system if a diskette is removed and replaced by another diskette, unless the new diskette is to be read only.

- (18) Load and test the debugger by typing

DDT

(see the document "CP/M Dynamic Debugging Tool (DDT)" for operating information and examples). Take time to familiarize yourself with DDT; it will be your best friend in later steps.

- (19) Before making further CBIOS modifications, practice using the editor (see the ED user's guide), and assembler (see the ASM user's guide). Then

recode and test the GETSYS, PUTSYS, and CBIOS programs using ED, ASM, and DDT. Code and test a COPY program which does a sector-to-sector copy from one diskette to another to obtain back-up copies of the original diskette (NOTE: read your CP/M Licensing Agreement; it specifies your legal responsibilities when copying the CP/M system). Place the copyright notice

Copyright (c) 1978
Digital Research

on each copy which is made with your COPY program.

(20) Modify your CBIOS to include the extra functions for punches, readers, signon messages, and so-forth, and add the facilities for additional drives, if they exist on your system. You can make these changes with the GETSYS and PUTSYS programs which you have developed, or you can refer to the following section, which outlines CP/M facilities which will aid you in the regeneration process.

You now have a good copy of the customized CP/M system. Note that although the CBIOS portion of CP/M which you have developed belongs to you, the modified version of CP/M which you have created can be copied for your use only (again, read your Licensing Agreement) and cannot be legally copied for anyone else's use.

It should be noted that your system remains file-compatible with all other CP/M systems, which allows transfer of non-proprietary software between users of CP/M.

3. SECOND LEVEL SYSTEM GENERATION

Now that you have the CP/M system running, you will want to configure CP/M for your memory size. In general, you will first get a memory image of CP/M with the "MOVCPM" program (system relocater) and place this memory image onto a named disk file. The disk file can then be loaded, examined, patched, and replaced using the editor, assembler, debugger, and system generation program. For further details on the operation of these programs, see the "Guide to CP/M Features and Facilities" manual.

To get the memory image of CP/M into the TPA configured for the desired memory size, give the command:

```
MOVCPM xx *
```

where "xx" is the memory size in decimal K bytes (e.g., 32 for 32K). The response will be:

```
CONSTRUCTING xxK CP/M VERS 1.4  
READY FOR "SYSGEN" OR  
"SAVE 32 CPMxx.COM"
```

At this point, the image of CP/M in the TPA is configured for the desired memory size. The memory image is at location 0900H through 207FH (i.e., the BOOT is at 0900H, the CCP is at 980H, and the BIOS is at 1E80H). Note that the memory image has the standard MDS-800 BIOS and BOOT on it. It is now necessary to save the memory image in a file so that you can patch your CBIOS and CBOOT into it:

```
SAVE 32 CPMxx.COM
```

Save 20H = 32 pages of memory

The memory image created by the "MOVCPM" program is offset by a negative bias so that it loads into the free area of the TPA, and thus does not interfere with the operation of CP/M in higher memory. This memory image can be subsequently loaded under DDT and examined or changed in preparation for a new generation of the system. DDT is loaded with the memory image by typing:

```
DDT CPMxx.COM
```

Load DDT, then read the CPM image

DDT should respond with

```
NEXT PC  
2100 0100
```

You can then use the display (D) and disassembly (L) commands to examine portions of the memory image between 900H and 207FH. Note, however, that to find any particular address within the memory image, you must apply the negative bias to the CP/M address to find the actual address. Track 00, sector 01 is loaded to location 900H (you should find the cold start loader at

900H to 97FH), track 00, sector 02 is loaded into 980H (this is the base of the CCP), and so-forth through the entire CP/M system load. In a 16K system, for example, the CCP resides at the CP/M address 2900H, but is placed into memory at 980H by the SYSGEN program. Thus, the negative bias, denoted by n, satisfies

$$2900H + n = 980H, \text{ or } n = 980H - 2900H$$

Assuming two's complement arithmetic, $n = 0E080H$, which can be checked by

$$2900H + 0E080H = 10980H = 0980H \text{ (ignoring high-order overflow).}$$

Note that for larger systems, n satisfies

$$\begin{aligned} (2900H+b) + n &= 980H, \text{ or} \\ n &= 980H - (2900H + b), \text{ or} \\ n &= 0E080H - b. \end{aligned}$$

The value of n for common CP/M systems is given below

memory size	bias b	negative offset n
16K	0000H	0E080H - 0000H = 0E080H
24K	2000H	0E080H - 2000H = 0C080H
32K	4000H	0E080H - 4000H = 0A080H
40K	6000H	0E080H - 6000H = 8080H
48K	8000H	0E080H - 8000H = 6080H
56K	0A000H	0E080H - 0A000H = 4080H
62K	0B800H	0E080H - 0B800H = 2880H
64K	0C000H	0E080H - 0C000H = 2080H

Assume, for example, that you want to locate the address x within the memory image loaded under DDT in a 16K system. First type

Hx,n

Hexadecimal sum and difference

and DDT will respond with the value of x+n (sum) and x-n (difference). The first number printed by DDT will be the actual memory address in the image where the data or code will be found. The input

H2900,E080

for example, will produce 980H as the sum, which is where the CCP is located in the memory image under DDT.

Use the L command to disassemble portions of your CBIOS located at $(3E00H+b)+n$ which, when you use the H command, produces an actual address of 1E80H. The disassembly command would thus be

L1E80

Terminate DDT by typing a control-C or "G0" in order to prepare the patch program. Your CBIOS and BOOT can be modified using the editor and assembled using ASM, producing files called CBIOS.HEX and BOOT.HEX which contain the machine code for CBIOS and BOOT in Intel hex format. In order to integrate your new modules, return to DDT by typing

DDT CPMxx.COM

Start DDT and load the CPMxx image

It is now necessary to patch in your CBOOT and CBIOS routines. The BOOT resides at location 0900H in the memory image. If the actual load address is 'x', then to calculate the bias (m) use the command:

H900,x

Subtract load address from
target address.

The second number typed in response to the command is the desired bias (m). For example, if your BOOT executes at 0080H, the command:

H900,80

will reply

0980 0880

Sum and difference in hex.

Therefore, the bias "m" would be 0880H. To read the BOOT in, give the command:

ICBOOT.HEX

Input file CBOOT.HEX

Then:

Rm

Read CBOOT with a bias of
m (=900H-x)

You may now examine your CBOOT with:

L900

We are now ready to replace the CBIOS. Examine the area at 1E80H where the previous version of the CBIOS resides. Then type

ICBIOS.HEX

Ready the hex file for loading

Assume that your CBIOS is being integrated into a 16K CP/M system, and thus is based at location 3E00H. In order to properly locate the CBIOS in the memory image under DDT, we must apply the negative bias n for a 16K system when loading the hex file. This is accomplished by typing

RE080

Read the file with bias 0E080H

Upon completion of the read, re-examine the area where the CBIOS has been loaded (use an "LLE80" command), to ensure that it was loaded properly. When you are satisfied that the patch has been made, return from DDT using a control-C or "G0" command.

Now use SYSGEN to place the patched memory image back onto a diskette (use a test diskette until you are sure of your patch), as shown in the following interaction:

SYSGEN	Start the SYSGEN program
SYSGEN VERSION 1.4	Sign-on message from SYSGEN
SOURCE DRIVE NAME (OR RETURN TO SKIP)	Respond with a carriage return to skip the CP/M read operation since the system is already in memory.
DESTINATION DRIVE NAME (OR RETURN TO REBOOT)	Respond with B to write the new system to the diskette in drive B.
DESTINATION ON B, THEN TYPE RETURN	Hit the return key to perform the actual write.
FUNCTION COMPLETE	
DESTINATION DRIVE NAME (OR RETURN TO REBOOT)	Respond with a carriage return to reboot.

Place the test diskette on drive B (if you are operating with a single-drive system, answer "A" rather than "B" to the DESTINATION request; then remove your diskette, and replace it with the test diskette), and type a return. The system will be replaced on the test diskette. Test the new CP/M system by placing the test diskette in drive A and cold-starting.

Write the Digital Research copyright notice on the diskette, as specified in your Licensing Agreement:

Copyright (c), 1978
Digital Research

4. SAMPLE GETSYS AND PUTSYS PROGRAMS

The following program provides a framework for the GETSYS and PUTSYS programs referenced in Section 2. The READSEC and WRITESEC subroutines must be inserted by the user to read and write the specific sectors.

```

; GETSYS PROGRAM - READ TRACKS 0 AND 1 TO MEMORY AT 2880H
; REGISTER          USE
; A                (SCRATCH REGISTER)
; B                TRACK COUNT (0, 1)
; C                SECTOR COUNT (1,2,...,26)
; DE              (SCRATCH REGISTER PAIR)
; HL              LOAD ADDRESS
; SP              SET TO STACK ADDRESS
;
START: LXI    SP,2880H    ;SET STACK POINTER TO SCRATCH AREA
      LXI    H, 2880H    ;SET BASE LOAD ADDRESS
      MVI    B, 0        ;START WITH TRACK 0
RDTRK:                ;READ NEXT TRACK (INITIALLY 0)
      MVI    C,1        ;READ STARTING WITH SECTOR 1
RDSEC:                ;READ NEXT SECTOR
      CALL   READSEC    ;USER-SUPPLIED SUBROUTINE
      LXI    D,128      ;MOVE LOAD ADDRESS TO NEXT 1/2 PAGE
      DAD    D          ;HL = HL + 128
      INR    C          ;SECTOR = SECTOR + 1
      MOV    A,C        ;CHECK FOR END OF TRACK
      CPI    27
      JC     RDSEC      ;CARRY GENERATED IF SECTOR < 27
;
; ARRIVE HERE AT END OF TRACK, MOVE TO NEXT TRACK
      INR    B
      MOV    A,B        ;TEST FOR LAST TRACK
      CPI    2
      JC     RDTRK      ;CARRY GENERATED IF TRACK < 2
;
; ARRIVE HERE AT END OF LOAD, HALT FOR NOW
      HLT
;
; USER-SUPPLIED SUBROUTINE TO READ THE DISK
READSEC:
; ENTER WITH TRACK NUMBER IN REGISTER B,
; SECTOR NUMBER IN REGISTER C, AND
; ADDRESS TO FILL IN HL
;
      PUSH   B          ;SAVE B AND C REGISTERS
      PUSH   H          ;SAVE HL REGISTERS
      .....
      Perform disk read at this point, branch to
      label START if an error occurs

```

```

.....
POP    H           ;RECOVER HL
POP    B           ;RECOVER B AND C REGISTERS
RET                    ;BACK TO MAIN PROGRAM

```

```

END      START

```

Note that this program is assembled with an assumed origin of 0100. and listed in Appendix D for reference purposes. The hexadecimal operation codes which are listed on the left may be useful if the program has to be entered through your machine's front panel switches.

The PUTSYS program can be constructed from GETSYS by changing only a few operations in the GETSYS program given above, as shown in Appendix E. The register pair HL becomes the dump address (next address to write), and operations upon these registers do not change within the program. The READSEC subroutine is replaced by a WRITESEC subroutine which performs the opposite function: data from address HL is written to the track given by register B and the sector given by register C. It is often useful to combine GETSYS and PUTSYS into a single program during the test and development phase, as shown in Appendix E.

5. DISKETTE ORGANIZATION

The sector allocation for the standard distribution version of CP/M is given here for reference purposes. The first sector (see Figure 1) contains an optional software boot section. Disk controllers are often set up to bring track 0, sector 1 into memory at a specific location (often location 0000H). The program in this sector, called LBOOT, has the responsibility of bringing the remaining sectors into memory starting at location 2900H+b. If your controller does not have a built-in sector load, you can ignore the program in track 0, sector 1 and begin the load from track 0 sector 2 to location 2900H+b.

As an example, the Intel MDS-800 hardware cold start loader brings track 0, sector 1 into absolute address 3000H. Thus, the distribution version contains two very small programs in track 0, sector 1:

MBOOT - a storage move program which moves LBOOT into place following the cold start (Appendix A)

LBOOT - the cold start boot loader (Appendix B)

Upon MDS start-up, the 128 byte segment on track 0, sector 1 is brought into 3000H. The MBOOT program gets control, and moves the LBOOT program from location 301EH down to location 80H in memory, in order to get LBOOT out of the area where CP/M is loaded in a 16K system. Note that the MBOOT program would not be needed if the MDS loaded directly to 80H. In general, the LBOOT program could be located anywhere outside the CP/M load area, but is most often located in the area between 000H and 0FFH (below the TPA).

After the move, MBOOT transfers to LBOOT at 80H. LBOOT, in turn, loads the remainder of track 0 and the initialized portion of track 1 to memory, starting at 2900H+b. The user should note that MBOOT and LBOOT are of little use in a non-MDS environment, although it is useful to study them since some of their actions will have to be duplicated in your cold start loader.

Figure 1. Diskette Allocation

Track#	Sector#	Page#	Memory Address	CP/M Module name
00	01		(boot address)	Cold Start Loader
00	02	00	2900H+b	CCP
"	03	"	2980H+b	"
"	04	01	2A00H+b	"
"	05	"	2A80H+b	"
"	06	02	2B00H+b	"
"	07	"	2B80H+b	"
"	08	03	2C00H+b	"
"	09	"	2C80H+b	"

"	10	04	2D00H+b	"
"	11	"	2D80H+b	"
"	12	05	2E00H+b	"
"	13	"	2E80H+b	"
"	14	06	2F00H+b	"
"	15	"	2F80H+b	"
"	16	07	3000H+b	"
00	17	"	3080H+b	CCP
00	18	08	3100H+b	BDOS
"	19	"	3180H+b	"
"	20	09	3200H+b	"
"	21	"	3280H+b	"
"	22	10	3300H+b	"
"	23	"	3380H+b	"
"	24	11	3400H+b	"
"	25	"	3480H+b	"
"	26	12	3500H+b	"
01	01	"	3580H+b	"
"	02	13	3600H+b	"
"	03	"	3680H+b	"
"	04	14	3700H+b	"
"	05	"	3780H+b	"
"	06	15	3800H+b	"
"	07	"	3880H+b	"
"	08	16	3900H+b	"
"	09	"	3980H+b	"
"	10	17	3A00H+b	"
"	11	"	3A80H+b	"
"	12	18	3B00H+b	"
"	13	"	3B80H+b	"
"	14	19	3C00H+b	"
"	15	"	3C80H+b	"
"	16	20	3D00H+b	"
"	17	"	3D80H+b	BDOS
01	18	21	3E00H+b	BIOS
"	19	"	3E80H+b	"
"	20	22	3F00H+b	"
01	21	"	3F80H+b	BIOS
01	22-26			(not currently used)
02-76	01-26			(directory and data)

6. THE BIOS ENTRY POINTS

The entry points into the BIOS from the cold start loader and BDOS are detailed below. Entry to the BIOS is through a "jump vector" between locations 3E00H+b and 3E2CH+b, as shown below (see also Appendices, pages C-2 and D-1). The jump vector is a sequence of 15 jump instructions which send program control to the individual BIOS subroutines. The BIOS subroutines may be empty for certain functions (i.e., they may contain a single RET operation) during regeneration of CP/M, but the entries must be present in the jump vector.

It should be noted that there is a 16 byte area reserved in page zero (see Section 9) starting at location 40H, which is available as a "scratch" area in case the BIOS is implemented in ROM by the user. This scratch area is never accessed by any other CP/M subsystem during operation.

The jump vector at 3E00H+b takes the form shown below, where the individual jump addresses are given to the left:

3E00H+b	JMP BOOT	;ARRIVE HERE FROM COLD START LOAD
3E03H+b	JMP WBOOT	;ARRIVE HERE FOR WARM START
3E06H+b	JMP CONST	;CHECK FOR CONSOLE CHAR READY
3E09H+b	JMP CONIN	;READ CONSOLE CHARACTER IN
3E0CH+b	JMP CONOUT	;WRITE CONSOLE CHARACTER OUT
3E0FH+b	JMP LIST	;WRITE LISTING CHARACTER OUT
3E12H+b	JMP PUNCH	;WRITE CHARACTER TO PUNCH DEVICE
3E15H+b	JMP READER	;READ READER DEVICE
3E18H+b	JMP HOME	;MOVE TO TRACK 00 ON SELECTED DISK
3E1BH+b	JMP SELDSK	;SELECT DISK DRIVE
3E1EH+b	JMP SETTRK	;SET TRACK NUMBER
3E21H+b	JMP SETSEC	;SET SECTOR NUMBER
3E24H+b	JMP SETDMA	;SET DMA ADDRESS
3E27H+b	JMP READ	;READ SELECTED SECTOR
3E2AH+b	JMP WRITE	;WRITE SELECTED SECTOR

Each jump address corresponds to a particular subroutine which performs the specific function, as outlined below. There are three major divisions in the jump table: (1) the system (re)initialization which results from calls on BOOT and WBOOT, (2) simple character I/O performed by calls on CONST, CONIN, CONOUT, LIST, PUNCH, and READER, and (3) diskette I/O performed by calls on HOME, SELDSK, SETTRK, SETSEC, SETDMA, READ, and WRITE.

All simple character I/O operations are assumed to be performed in ASCII, upper and lower case, with high order (parity bit) set to zero. An end-of-file condition is given by an ASCII control-z (1AH). Peripheral devices are seen by CP/M as "logical" devices, and are assigned to physical devices within the BIOS. In order to operate, the BDOS needs only the CONST, CONIN, and CONOUT subroutines (LIST, PUNCH, and READER are used by PIP, but not by the BDOS). Thus, the initial version of CBIOS may have empty

subroutines for the remaining ASCII devices. The characteristics of each device are

CONSOLE	The principal interactive console which communicates with the operator, accessed through CONST, CONIN, and CONOUT. Typically, the CONSOLE is a device such as a CRT or Teletype.
LIST	The principal listing device, if it exists on your system, which is usually a hard-copy device, such as a printer or Teletype.
PUNCH	The principal tape punching device, if it exists, which is normally a high-speed paper tape punch or Teletype.
READER	The principal tape reading device, such as a simple optical reader or Teletype.

Note that a single peripheral can be assigned as the LIST, PUNCH, and READER device simultaneously. If no peripheral device is assigned as the LIST, PUNCH, or READER device, the CBIOS created by the user should give an appropriate error message so that the system does not "hang" if the device is accessed by PIP or some other user program. Alternately, the PUNCH and LIST routines can simply return, and the READER routine can return with a LAH (ctl-Z) in reg A to indicate immediate end-of-file.

For added flexibility, the user can optionally implement the "IOBYTE" function which allows reassignment of physical and logical devices. The IOBYTE function creates a mapping of logical to physical devices which can be altered during CP/M processing (see the STAT command). The definition of the IOBYTE function corresponds to the Intel standard as follows: a single location in memory (currently location 0003H) is maintained, called IOBYTE, which defines the logical to physical device mapping which is in effect at a particular time. The mapping is performed by splitting the IOBYTE into four distinct fields of two bits each, called the CONSOLE, READER, PUNCH, and LIST fields, as shown below:

	most significant				least significant
IOBYTE AT 0003H	LIST	PUNCH	READER	CONSOLE	
	bits 6,7	bits 4,5	bits 2,3	bits 0,1	

The value in each field can be in the range 0-3, defining the assigned source or destination of each logical device. The values which can be assigned to each field are given below

CONSOLE field (bits 0,1)

- 0 - console is assigned to the console printer device (TTY:)
- 1 - console is assigned to the CRT device (CRT:)
- 2 - batch mode: use the READER as the CONSOLE input,
and the LIST device as the CONSOLE output (BAT:)
- 3 - user-defined console device (UC1:)

READER field (bits 2,3)

- 0 - READER is the Teletype device (TTY:)
- 1 - READER is the high-speed reader device (PTR:)
- 2 - user-defined reader # 1 (UR1:)
- 3 - user-defined reader # 2 (UR2:)

PUNCH field (bits 4,5)

- 0 - PUNCH is the Teletype device (TTY:)
- 1 - PUNCH is the high speed punch device (PTP:)
- 2 - user-defined punch # 1 (UP1:)
- 3 - user-defined punch # 2 (UP2:)

LIST field (bits 6,7)

- 0 - LIST is the Teletype device (TTY:)
- 1 - LIST is the CRT device (CRT:)
- 2 - LIST is the line printer device (LPT:)
- 3 - user-defined list device (UL1:)

Note again that the implementation of the IOBYTE is optional, and affects only the organization of your CBIOS. No CP/M systems use the IOBYTE (although they tolerate the existence of the IOBYTE at location 0003H), except for PIP which allows access to the physical devices, and STAT which allows logical-physical assignments to be made and/or displayed (for more information, see the "CP/M Features and Facilities Guide"). In any case, the IOBYTE implementation should be omitted until your basic CBIOS is fully implemented and tested; then add the IOBYTE to increase your facilities.

Disk I/O is always performed through a sequence of calls on the various disk access subroutines. These set up the disk number to access, the track and sector on a particular disk, and the direct memory access (DMA) address involved in the I/O operation. After all these parameters have been set up, a call is made to the READ or WRITE function to perform the actual I/O operation. Note that there is often a single call to SELDSK to select a disk drive, followed by a number of read or write operations to the selected disk, before selecting another drive for subsequent operations. Similarly, there may be a single call to set the DMA address, followed by several calls which read or write from the selected DMA address, before the DMA address is changed. The track and sector subroutines are always called before the READ or WRITE operations are performed. Note that the READ and WRITE routines should perform several re-attempts (10 is a good number) before reporting the error condition to the BDOS. If the error condition is returned to the BDOS, it will report the error to the user. The HOME subroutine may or may not actually perform the track 00 seek, depending upon your controller

characteristics; the important point is that track 00 has been selected for the next operation, and is often treated in exactly the same manner as SETTRK with a parameter of 00.

The exact responsibilities of each entry point subroutine are given below:

BOOT The BOOT entry point gets control from the cold start loader and is responsible for basic system initialization, including sending a signon message (which can be omitted in the first version). If the IOBYTE function is implemented, it must be set at this point. The various system parameters which are set by the WBOOT entry point must be initialized, and control is transferred to the CCP at 2900H+b for further processing. Note that reg C must be set to zero to select drive A.

WBOOT The WBOOT entry point gets control when a warm start occurs. A warm start is performed whenever a user program branches to location 0000H, or when the CPU is reset from the front panel. The CP/M system must be loaded from the first two tracks of drive A up to, but not including, the BIOS (or CBIOS, if you have completed your patch). System parameters must be initialized as shown below:

location 0,1,2	Set to JMP WBOOT for warm starts (0000H: JMP 3E03H+b).
location 3	Set initial value of IOBYTE, if implemented in your CBIOS.
location 5,6,7	Set to JMP BDOS, which is the primary entry point to CP/M for transient programs (0005H: JMP 3106H+b).

(See Section 9 for complete details of page zero use.)
Upon completion of the initialization, the WBOOT program must branch to the CCP at 2900H+b to (re)start the system. Upon entry to the CCP, register C is set to the drive to select after system initialization.

CONST Sample the status of the currently assigned console device; return 0FFH in register A if a character is ready to read and 00H in register A if no console characters are ready.

CONIN Read the next console character into register A, and set the high-order (parity bit). If no console character is ready, wait until a character is typed before returning.

CONOUT Send the character from register C to the console output device. The character is in ASCII, with high-order (parity) bit set to zero. You may want to include a time-out on a line

feed or carriage return, if your console device requires some time interval at the end of the line (such as a TI Silent 700 terminal). You can, if you wish, filter out control characters which cause your console device to react in a strange way (a control-z causes the Lear Seigler terminal to clear the screen, for example).

- LIST Send the character from register C to the currently assigned listing device. The character is in ASCII with zero parity.
- PUNCH Send the character from register C to the currently assigned punch device. The character is in ASCII with zero parity.
- READER Read the next character from the currently assigned reader device into register A with zero parity (high-order bit must be zero), an end-of-file condition is reported by returning an ASCII control-z (1AH).
- HOME Return the disk head of the currently selected disk (initially disk A) to the track 00 position. If your controller allows access to the track 0 flag from the drive, step the head until the track 0 flag is detected. If your controller does not support this feature, you can translate the HOME call into a call on SETTRK with a parameter of 0.
- SELDSK Select the disk drive given by register C for further operations, where register C contains 0 for drive A, 1 for drive B, 2 for drive C, and 3 for drive D. (The standard CP/M distribution version supports a maximum of four drives). If your system has less than 4 drives, you may wish to give an error message at the console, and terminate execution. It is advisable to postpone the actual disk select operation until an I/O function (seek, read or write) is actually performed, since disk selects often occur without ultimately performing any disk I/O, and many controllers will unload the head of the current disk before selecting the new drive. This would cause an excessive amount of noise and disk wear.
- SETTRK Register C contains the track number for subsequent disk accesses on the currently selected drive. You can choose to seek the selected track at this time, or delay the seek until the next read or write actually occurs. Register C can take on values in the range 0-76 corresponding to valid track numbers.
- SETSEC Register C contains the sector number (1 through 26) for subsequent disk accesses on the currently selected drive. You can choose to send this information to the controller at this point, or instead delay sector selection until a read or write operation occurs.

SETDMA

Registers B and C (high-order 8 bits in B, low-order 8 bits in C) contain the DMA (Direct Memory Access) address for subsequent read or write operations. For example, if B = 00H and C = 80H when SETDMA is called, then all subsequent read operations read their data into 80H through 0FFH, and all subsequent write operations get their data from 80H through 0FFH, until the next call to SETDMA occurs. The initial DMA address is assumed to be 80H. Note that the controller need not actually support direct memory access. If, for example, all data is received and sent through I/O ports, the CBIOS which you construct will use the 128-byte area starting at the selected DMA address for the memory buffer during the following read or write operations.

READ

Assuming the drive has been selected, the track has been set, the sector has been set, and the DMA address has been specified, the READ subroutine attempts to read one sector based upon these parameters, and returns the following error codes in register A:

0	no errors occurred
1	non-recoverable error condition occurred

Currently, CP/M responds only to a zero or non-zero value as the return code. That is, if the value in register A is 0 then CP/M assumes that the disk operation completed properly. If an error occurs, however, the CBIOS should attempt at least 10 re-tries to see if the error is recoverable. When an error is reported the BDOS will print the message "BDOS ERR ON x: BAD SECTOR." The operator then has the option of typing <cr> to ignore the error, or control-C to abort.

WRITE

Write the data from the currently selected DMA address to the currently selected drive, track, and sector. The data should be marked as "non deleted data" to maintain compatibility with other CP/M systems. The error codes given in the READ command are returned in register A, with error recovery attempts as described above.

7. A SAMPLE BIOS

The program shown in Appendix D can serve as a basis for your first BIOS. The simplest functions are assumed in this BIOS, so that you can enter it through the front panel, if absolutely necessary. Note that the user must alter and insert code into the subroutines for CONST, CONIN, CONOUT, READ, WRITE, and WAPTIO. Storage is reserved for user-supplied code in these regions. The scratch area reserved in page zero (see Section 9) for the BIOS is used in this program, so that it could be implemented in ROM, if desired.

Once operational, this skeletal version can be enhanced to print the initial sign-on message and perform better error recovery. The subroutines for LIST, PUNCH, and READER can be filled-out, and the IOBYTE function can be implemented.

8. A SAMPLE COLD START LOADER

The program shown in Appendix E can serve as a basis for your cold start loader. The disk read function must be supplied by the user, and the program must be loaded somehow starting at location 0000. Note that space is reserved for your patch so that the total amount of storage required for the cold start loader is 128 bytes. Eventually, you will probably want to get this loader onto the first disk sector (track 0, sector 1) and cause your controller to load it into memory automatically upon system start-up. Alternatively, you may wish to place the cold start loader into ROM and place it above the CP/M system. In this case, it will be necessary to originate the program at a higher address and key-in a jump instruction at system start-up which branches to the loader. Subsequent warm starts will not require this key-in operation, since the entry point 'WBOOT' gets control, thus bringing the system in from disk automatically. Note also that the skeletal cold start loader has minimal error recovery, which may be enhanced on later versions.

9. RESERVED LOCATIONS IN PAGE ZERO

Main memory page zero, locations 00H through 0FFH, contains several segments of code and data which are used during CP/M processing. The code and data areas are given below for reference purposes.

Locations from to	Contents
0000H - 0002H	Contains a jump instruction to the warm start entry point at location 3E03H+b. This allows a simple programmed restart (JMP 0000H) or manual restart from the front panel.
0003H - 0003H	Contains the Intel standard IOBYTE, which is optionally included in the user's CBIOS, as described in Section 6.
0004H - 0004H	Current default drive number (0=A, 1=B, 2=C, 3=D).
0005H - 0007H	Contains a jump instruction to the BDOS, and serves two purposes: JMP 0005H provides the primary entry point to the BDOS, as described in the manual "CP/M Interface Guide," and LHL 0006H brings the address field of the instruction to the HL register pair. This value is the lowest address in memory used by CP/M (assuming the CCP is being overlayed). Note that the DDT program will change the address field to reflect the reduced memory size in debug mode.
0008H - 0027H	(interrupt locations 1 through 5 not used)
0030H - 0037H	(interrupt location 6, not currently used - reserved)
0038H - 003AH	Contains a jump instruction into the DDT program when running in debug mode for programmed breakpoints, but is not otherwise used by CP/M.
003BH - 003FH	(not currently used - reserved)
0040H - 004FH	16 byte area reserved for scratch by CBIOS, but is not used for any purpose in the distribution version of CP/M
0050H - 005BH	(not currently used - reserved)
005CH - 007CH	Default File Control Block produced for a transient program by the Console Command Processor.
007DH - 007FH	(not currently used - reserved)

0080H - 00FFH Default 128-byte disk buffer (also filled with the command line when a transient is loaded under the CCP).

Note that this information is setup for normal operation under the CP/M system, but can be overwritten by a transient program if the BDOS facilities are not required by the transient. If, for example, a particular program performs only simple I/O and must begin execution at location 0, it can be first loaded into the TPA, using normal CP/M facilities, with a small memory move program which gets control when loaded (the memory move program must get control from location 0100H, which is the assumed beginning of all transient programs). The move program can then proceed to move the entire memory image down to location 0, and pass control to the starting address of the memory load. Note that if the BIOS is overwritten, or if location 0 (containing the warm start entry point) is overwritten, then the programmer must bring the CP/M system back into memory with a cold start sequence.

10. NOTES FOR USERS OF CP/M VERSION 1.3

The only difference in memory layout between CP/M versions 1.3 and 1.4 is the location of the BDOS, which has been moved down one page (3100h+b instead of 3200h+b). Therefore, your present CBIOS must be changed to reflect this. Normally, the only change is found in the initialization of the jump instruction at location 5. This jump should now be JMP 3106H+b instead of JMP 3206H+b. Note that the CCP is one page shorter, offsetting the longer BDOS, so that the system load address (2900H+b) remains the same. CP/M 1.4 also supports four drives, and thus your CBIOS must account for a drive select value in the range 0-3. No other changes to CP/M affect the CBIOS organization.

APPENDIX A: THE MDS LOADER MOVE PROGRAM

```

;      MDS LOADER MOVE PROGRAM, PLACES COLD START BOOT AT BOOTB
;
3000      ORG      3000H      ;WE ARE LOADED HERE ON COLD START
0080 =    BOOTB    EQU      80H      ;START OF COLD BOOT PROGRAM
0080 =    BOOTL    EQU      80H      ;LENGTH OF BOOT
D900 =    MBIAS    EQU      900H-$    ;BIAS TO ADD DURING LOAD
0078 =    BASE     EQU      078H      ;'BASE' USED BY DISK CONTROLLER
0079 =    RTYPE    EQU      BASE+1    ;RESULT TYPE
007B =    RBYTE    EQU      BASE+3    ;RESULT TYPE

;
00FF =    BSW      EQU      0FFH      ;BOOT SWITCH
;
;      CLEAR DISK STATUS
3000 DB79      IN      RTYPE
3002 DB7B      IN      RBYTE
;
COLDSTART:
3004 DBFF      IN      BSW
3006 E602      ANI      2H      ;SWITCH ON?
3008 C20430     JNZ      COLDSTART
;
300B 211E30     LXI      H,BOOTV ;VIRTUAL BASE
300E 0680      MVI      B,BOOTL ;LENGTH OF BOOT
3010 118000     LXI      D,BOOTB ;DESTINATION OF BOOT
3013 7E        MOVE:    MOV      A,M
3014 12        STAX     D      ;TRANSFERRED ONE BYTE
3015 23        INX      H
3016 13        INX      D
3017 05        DCR      B
3018 C21330     JNZ      MOVE
301B C38000     JMP      BOOTB      ;TO BOOT SYSTEM
;
BOOTV: ;BOOT LOADER PLACE HERE AT SYSTEM GENERATION
089E =    LBIAS    EQU      $-80H+MBIAS ;COLD START BOOT BEGINS AT 80H
301E      END

```

APPENDIX B: THE MDS COLD START LOADER

```

;
; MDS COLD START LOADER FOR CP/M
; VERSION 1.4 JANUARY, 1978
;
0100 = BIAS EQU 100H ;BIAS FOR RELOCATION
0000 = FALSE EQU 0
FFFF = TRUE EQU NOT FALSE
0000 = TESTING EQU FALSE ;IF TRUE, THEN GO TO MON80 ON ERRORS
;
0100 = BDOSB EQU BIAS ;BASE OF DOS LOAD
0906 = BDOS EQU 806H+BIAS ;ENTRY TO DOS FOR CALLS
1800 = BDOSE EQU 1700H+BIAS ;END OF DOS LOAD
1600 = BOOT EQU 1500H+BIAS ;COLD START ENTRY POINT
1603 = RBOOT EQU BOOT+3 ;WARM START ENTRY POINT
;
0080 ORG 80H ;LOADED DOWN FROM HARDWARE BOOT AT 3000H
;
1700 = BDOSL EQU BDOSE-BDOSB
0002 = NTRKS EQU 2 ;NUMBER OF TRACKS TO READ
002E = BDOS EQU BDOSL/128 ;NUMBER OF SECTORS IN DOS
0019 = BDOS0 EQU 25 ;NUMBER OF BDOS SECTORS ON TRACK 0
0015 = BDOS1 EQU BDOS-BDOS0 ;NUMBER OF SECTORS ON TRACK 1
;
F800 = MON80 EQU 0F800H ;INTEL MONITOR BASE
FF0F = RMON80 EQU 0FF0FH ;RESTART LOCATION FOR MON80
0078 = BASE EQU 078H ;'BASE' USED BY CONTROLLER
0079 = RTYPE EQU BASE+1 ;RESULT TYPE
007B = RBYTE EQU BASE+3 ;RESULT BYTE
007F = RESET EQU BASE+7 ;RESET CONTROLLER
;
0078 = DSTAT EQU BASE ;DISK STATUS PORT
0079 = ILOW EQU BASE+1 ;LOW IOPB ADDRESS
007A = IHIGH EQU BASE+2 ;HIGH IOPB ADDRESS
0003 = RECAL EQU 3H ;RECALIBRATE SELECTED DRIVE
0004 = READF EQU 4H ;DISK READ FUNCTION
0100 = STACK EQU 100H ;USE END OF BOOT FOR STACK
;
RSTART:
0080 310001 LXI SP,STACK;IN CASE OF CALL TO MON80
; CLEAR THE CONTROLLER
0083 D37F OUT RESET ;LOGIC CLEARED
;
;
0085 0602 MVI B,NTRKS ;NUMBER OF TRACKS TO READ
0087 21B700 LXI H,IOPB0
;
START:

```

```

;
; READ FIRST/NEXT TRACK INTO BDOSB
008A 7D      MOV     A,L
008B D379    OUT     ILOW
008D 7C      MOV     A,H
008E D37A    OUT     IHIGH
0090 DB78    WAIT0:  IN      DSTAT
0092 E604    ANI     4
0094 CA9000  JZ      WAIT0

;
; CHECK DISK STATUS
0097 DB79    IN      RTYPE
0099 E603    ANI     11B
009B FE02    CPI     2

;
IF          TESTING
CNC         RMON80  ;GO TO MONITOR IF 11 OR 10
ENDIF
IF          NOT TESTING
009D D28000  JNC     RSTART ;RETRY THE LOAD
ENDIF

;
00A0 DB7B    IN      RBYTE ;I/O COMPLETE, CHECK STATUS
;
IF NOT READY, THEN GO TO MON80
00A2 17      RAL
00A3 DC0FFF  CC      RMON80 ;NOT READY BIT SET
00A6 1F      RAR      ;RESTORE
00A7 E61E    ANI     11110B ;OVERRUN/ADDR ERR/SEEK/CRC/XXXX

;
IF          TESTING
CNC         RMON80  ;GO TO MONITOR
ENDIF
IF          NOT TESTING
00A9 C28000  JNZ     RSTART ;RETRY THE LOAD
ENDIF

;
;
00AC 110700  LXI     D,IOPBL ;LENGTH OF IOPB
00AF 19      DAD     D      ;ADDRESSING NEXT IOPB
00B0 05      DCR     B      ;COUNT DOWN TRACKS
00B1 C28A00  JNZ     START

;
;
; JMP TO BOOT TO PRINT INITIAL MESSAGE, AND SET UP JMPs
00B4 C30016  JMP     BOOT

;
; PARAMETER BLOCKS
IOPB0: DB      80H      ;IOCW, NO UPDATE
        DB      READF   ;READ FUNCTION
        DB      BDOS0   ;# SECTORS TO READ ON TRACK 0

```

```

00BA 00          DB      0          ;TRACK 0
00BB 02          DB      2          ;START WITH SECTOR 2 ON TRACK 0
00BC 0001        DW      BDOSB      ;START AT BASE OF BDOS
0007 =          IOPBL EQU      $-IOPB0
;
00BE 80          IOPB1: DB      80H
00BF 04          DB      READF
00C0 15          DB      BDOS1      ;SECTORS TO READ ON TRACK 1
00C1 01          DB      1          ;TRACK 1
00C2 01          DB      1          ;SECTOR 1
00C3 800D        DW      BDOSB+BDOS0*128 ;BASE OF SECOND READ
;
00C5            END

```

APPENDIX C: THE MDS BASIC I/O SYSTEM (BIOS)

```

;      MDS I/O DRIVERS FOR CP/M
;      (FOUR DRIVE SINGLE DENSITY VERSION)
;      VERSION 1.4 JANUARY, 1978
;
000E =  VERS    EQU    14      ;VERSION 1.4
;
;      COPYRIGHT (C) 1978
;      DIGITAL RESEARCH
;      BOX 579, PACIFIC GROVE
;      CALIFORNIA, 93950
;
;
FFFF =  TRUE    EQU    0FFFFH ;VALUE OF "TRUE"
0000 =  FALSE   EQU    NOT TRUE ;"FALSE"
FFFF =  SAMPLE  EQU    TRUE   ;TRUE IF SAMPLE BIOS
;
;      IF      SAMPLE
2900 =  BIAS    EQU    2900H  ;SAMPLE PROGRAM IN 16K SYSTEM
;      ENDIF
;      IF      NOT SAMPLE
BIAS    EQU    0000H  ;GENERATE RELOCATABLE CP/M SYSTEM
;      ENDIF
;
3E00 =  PATCH   EQU    1500H+BIAS
;
3E00    ORG     PATCH
2900 =  CPMB    EQU    000H+BIAS ;BASE OF CPM CONSOLE PROCESSOR
3106 =  BDOS    EQU    806H+BIAS ;BASIC DOS (RESIDENT PORTION)
1500 =  CPML    EQU    $-CPMB ;LENGTH (IN BYTES) OF CPM SYSTEM
002A =  NSECTS  EQU    CPML/128 ;NUMBER OF SECTORS TO LOAD
0002 =  OFFSET  EQU    2 ;NUMBER OF DISK TRACKS USED BY CP/M
0004 =  CDISK   EQU    0004H ;ADDRESS OF LAST LOGGED DISK ON WARM START
0080 =  BUFF    EQU    0080H ;DEFAULT BUFFER ADDRESS
000A =  RETRY   EQU    10 ;MAX RETRIES ON DISK I/O BEFORE ERROR
;
;      PERFORM FOLLOWING FUNCTIONS
;      BOOT    COLD START
;      WBOOT   WARM START (SAVE I/O BYTE)
;      (BOOT AND WBOOT ARE THE SAME FOR MDS)
;      CONST   CONSOLE STATUS
;      REG-A = 00 IF NO CHARACTER READY
;      REG-A = FF IF CHARACTER READY
;      CONIN   CONSOLE CHARACTER IN (RESULT IN REG-A)
;      CONOUT  CONSOLE CHARACTER OUT (CHAR IN REG-C)
;      LIST    LIST OUT (CHAR IN REG-C)
;      PUNCH   PUNCH OUT (CHAR IN REG-C)
;      READER  PAPER TAPE READER IN (RESULT TO REG-A)
;      HOME    MOVE TO TRACK 00

```

```

;
; (THE FOLLOWING CALLS SET-UP THE IO PARAMETER BLOCK FOR THE
; MDS, WHICH IS USED TO PERFORM SUBSEQUENT READS AND WRITES)
; SELDSK SELECT DISK GIVEN BY REG-C (0,1,2...)
; SETTRK SET TRACK ADDRESS (0,...76) FOR SUBSEQUENT READ/WRITE
; SETSEC SET SECTOR ADDRESS (1,...,26) FOR SUBSEQUENT READ/WRITE
; SETDMA SET SUBSEQUENT DMA ADDRESS (INITIALLY 80H)
;
; (READ AND WRITE ASSUME PREVIOUS CALLS TO SET UP THE IO PARAMETERS)
; READ READ TRACK/SECTOR TO PRESET DMA ADDRESS
; WRITE WRITE TRACK/SECTOR FROM PRESET DMA ADDRESS
;
; JUMP VECTOR FOR INDIVIDUAL ROUTINES
3E00 C3443E JMP BOOT
3E03 C3543E WBOOTE: JMP WBOOT
3E06 C3F23E JMP CONST
3E09 C3F53E JMP CONIN
3E0C C3FB3E JMP CONOUT
3E0F C3FE3E JMP LIST
3E12 C3013F JMP PUNCH
3E15 C3043F JMP READER
3E18 C3073F JMP HOME
3E1B C30C3F JMP SELDSK
3E1E C32A3F JMP SETTRK
3E21 C32F3F JMP SETSEC
3E24 C3343F JMP SETDMA
3E27 C33A3F JMP READ
3E2A C3433F JMP WRITE
;
;
; END OF CONTROLLER - INDEPENDENT CODE, THE REMAINING SUBROUTINES
; ARE TAILORED TO THE PARTICULAR OPERATING ENVIRONMENT, AND MUST
; BE ALTERED FOR ANY SYSTEM WHICH DIFFERS FROM THE INTEL MDS.
;
; THE FOLLOWING CODE ASSUMES THE MDS MONITOR EXISTS AT 0F800H
; AND USES THE I/O SUBROUTINES WITHIN THE MONITOR
;
; WE ALSO ASSUME THE MDS SYSTEM HAS FOUR DISK DRIVES
0004 = NDISKS EQU 4 ;NUMBER OF DRIVES AVAILABLE
00FD = REVRT EQU 0FDH ;INTERRUPT REVERT PORT
00FC = INTC EQU 0FCH ;INTERRUPT MASK PORT
00F3 = ICON EQU 0F3H ;INTERRUPT CONTROL PORT
007E = INTE EQU 0111$1110B ;ENABLE RST 0(WARM BOOT), RST 7 (MONITOR)
;
; MDS MONITOR EQUATES
F800 = MON80 EQU 0F800H ;MDS MONITOR
FF0F = RMON80 EQU 0FF0FH ;RESTART MON80 (BOOT ERROR)
F803 = CI EQU 0F803H ;CONSOLE CHARACTER TO REG-A
F806 = RI EQU 0F806H ;READER IN TO REG-A
F809 = CO EQU 0F809H ;CONSOLE CHAR FROM C TO CONSOLE OUT

```

```

F80C =      PO      EQU      0F80CH ;PUNCH CHAR FROM C TO PUNCH DEVICE
F80F =      LO      EQU      0F80FH ;LIST FROM C TO LIST DEVICE
F812 =      CSTS    EQU      0F812H ;CONSOLE STATUS 00/FF TO REGISTER A
;
;      DISK PORTS AND COMMANDS
0078 =      BASE    EQU      78H    ;BASE OF DISK COMMAND IO PORTS
0078 =      DSTAT   EQU      BASE   ;DISK STATUS (INPUT)
0079 =      RTYPE   EQU      BASE+1 ;RESULT TYPE (INPUT)
007B =      RBYTE   EQU      BASE+3 ;RESULT BYTE (INPUT)
;
0079 =      ILOW    EQU      BASE+1 ;IOPB LOW ADDRESS (OUTPUT)
007A =      IHIGH   EQU      BASE+2 ;IOPB HIGH ADDRESS (OUTPUT)
;
0004 =      READF   EQU      4H     ;READ FUNCTION
0006 =      WRITEF  EQU      6H     ;WRITE FUNCTION
0003 =      RECAL   EQU      3H     ;RECALIBRATE DRIVE
0004 =      IORDY   EQU      4H     ;I/O FINISHED MASK
000D =      CR      EQU      0DH    ;CARRIAGE RETURN
000A =      LF      EQU      0AH    ;LINE FEED
;
SIGNON: ;SIGNON MESSAGE: XXK CP/M VERS Y.Y
3E2D 0D0A0A      DB      CR,LF,LF
                  IF      SAMPLE
3E30 3136         DB      '16'      ;16K EXAMPLE BIOS
                  ENDIF
                  IF      NOT SAMPLE
                  DB      '00'      ;MEMORY SIZE FILLED BY RELOCATOR
                  ENDIF
3E32 4B2043502F   DB      'K CP/M VERS '
3E3E 312E34       DB      VERS/10+'0','.',',VERS MOD 10+'0'
3E41 0D0A00       DB      CR,LF,0
;
BOOT: ;PRINT SIGNON MESSAGE AND GO TO CCP
;      (NOTE: MDS BOOT INITIALIZED IOBYTE AT 0003H)
3E44 310001       LXI      SP,BUFF+80H
3E47 212D3E       LXI      H,SIGNON
3E4A CD4C3F       CALL     PRMSG    ;PRINT MESSAGE
3E4D AF          XRA      A        ;CLEAR ACCUMULATOR
3E4E 320400       STA      CDISK    ;SET INITIALLY TO DISK A
3E51 C3A03E       JMP      GOCPM    ;GO TO CP/M
;
;
WBOOT:; LOADER ON TRACK 0, SECTOR 1, WHICH WILL BE SKIPPED FOR WARM
;      READ CP/M FROM DISK - ASSUMING THERE IS A 128 BYTE COLD START
;      START.
;
3E54 318000       LXI      SP,BUFF ;USING DMA - THUS 80 THRU FF AVAILABLE FOR STACK
;
3E57 0E0A        MVI      C,RETRY ;MAX RETRIES
3E59 C5          PUSH     B

```

```

WBOOT0: ;ENTER HERE ON ERROR RETRIES
3E5A 010029 LXI B,CPMB ;SET DMA ADDRESS TO START OF DISK SYSTEM
3E5D CD343F CALL SETDMA
3E60 0E00 MVI C,0 ;BOOT FROM DRIVE 0
3E62 CD0C3F CALL SELDSK
3E65 0E00 MVI C,0
3E67 CD2A3F CALL SETTRK ;START WITH TRACK 0
3E6A 0E02 MVI C,2 ;START READING SECTOR 2
3E6C CD2F3F CALL SETSEC

;
; READ SECTORS, COUNT NSECTS TO ZERO
3E6F C1 POP B ;10-ERROR COUNT
3E70 062A MVI B,NSECTS
RDSEC: ;READ NEXT SECTOR
3E72 C5 PUSH B ;SAVE SECTOR COUNT
3E73 CD3A3F CALL READ
3E76 C2DA3E JNZ BOOTERR ;RETRY IF ERRORS OCCUR
3E79 2AE53F LHLD IOD ;INCREMENT DMA ADDRESS
3E7C 118000 LXI D,128 ;SECTOR SIZE
3E7F 19 DAD D ;INCREMENTED DMA ADDRESS IN HL
3E80 44 MOV B,H
3E81 4D MOV C,L ;READY FOR CALL TO SET DMA
3E82 CD343F CALL SETDMA
3E85 3AE43F LDA IOS ;SECTOR NUMBER JUST READ
3E88 FE1A CPI 26 ;READ LAST SECTOR?
3E8A DA963E JC RD1
; MUST BE SECTOR 26, ZERO AND GO TO NEXT TRACK
3E8D 3AE33F LDA IOT ;GET TRACK TO REGISTER A
3E90 3C INR A
3E91 4F MOV C,A ;READY FOR CALL
3E92 CD2A3F CALL SETTRK
3E95 AF XRA A ;CLEAR SECTOR NUMBER
3E96 3C RD1: INR A ;TO NEXT SECTOR
3E97 4F MOV C,A ;READY FOR CALL
3E98 CD2F3F CALL SETSEC
3E9B C1 POP B ;RECALL SECTOR COUNT
3E9C 05 DCR B ;DONE?
3E9D C2723E JNZ RDSEC

;
; DONE WITH THE LOAD, RESET DEFAULT BUFFER ADDRESS
GOCPM: ;(ENTER HERE FROM COLD START BOOT)
; ENABLE RST0 AND RST7
3EA0 F3 DI
3EA1 3E12 MVI A,12H ;INITIALIZE COMMAND
3EA3 D3FD OUT REVRT
3EA5 AF XRA A
3EA6 D3FC OUT INTC ;CLEARED
3EA8 3E7E MVI A,INTE ;RST0 AND RST7 BITS ON
3EAA D3FC OUT INTC
3EAC AF XRA A

```

```

3EAD D3F3          OUT      ICON      ;INTERRUPT CONTROL
;
;
3EAF 018000        LXI      B,BUFF
3EB2 CD343F        CALL     SETDMA
;
;
; RESET MONITOR ENTRY POINTS
3EB5 3EC3          MVI      A,JMP
3EB7 320000        STA      0
3EBA 21033E        LXI      H,WBOOT
3EBD 220100        SHLD     1          ;JMP WBOOT AT LOCATION 00
3EC0 320500        STA      5
3EC3 210631        LXI      H,BDOS
3EC6 220600        SHLD     6          ;JMP BDOS AT LOCATION 5
3EC9 323800        STA      7*8        ;JMP TO MON80 (MAY HAVE BEEN CHANGED BY DDT)
3ECC 2100F8        LXI      H,MON80
3ECF 223900        SHLD     7*8+1
;
; LEAVE IOBYTE SET
;
3ED2 3A0400        LDA      CDISK      ;LAST LOGGED DISK NUMBER
3ED5 4F            MOV      C,A        ;SEND TO CCP TO LOG IT IN
3ED6 FB            EI
3ED7 C30029        JMP      CPMB
;
;
; ERROR CONDITION OCCURRED, PRINT MESSAGE AND RETRY
BOOTERR:
3EDA C1            POP      B          ;RECALL COUNTS
3EDB 0D            DCR      C
3EDC CAE33E        JZ       BOOTER0
;
; TRY AGAIN
3EDF C5            PUSH     B
3EE0 C35A3E        JMP      WBOOT0
;
;
BOOTER0:
;
; OTHERWISE TOO MANY RETRIES
3EE3 21EC3E        LXI      H,BOOTMSG
3EE6 CD4C3F        CALL     PRMSG
3EE9 C30FFF        JMP      RMON80    ;MDS HARDWARE MONITOR
;
;
BOOTMSG:
3EEC 3F424F4F54    DB       '?BOOT',0
;
;
;
CONST: ;CONSOLE STATUS TO REG-A
;
; (EXACTLY THE SAME AS MDS CALL)
3EF2 C312F8        JMP      CSTS
;
;
CONIN: ;CONSOLE CHARACTER TO REG-A
3EF5 CD03F8        CALL     CI
3EF8 E67F          ANI      7FH        ;REMOVE PARITY BIT

```

```

3EFA C9          RET
;
;CONOUT: ;CONSOLE CHARACTER FROM C TO CONSOLE OUT
3EFB C309F8      JMP     CO
;
;LIST: ;LIST DEVICE OUT
;          (EXACTLY THE SAME AS MDS CALL)
3EFE C30FF8      JMP     LO
;
;PUNCH: ;PUNCH DEVICE OUT
;          (EXACTLY THE SAME AS MDS CALL)
3F01 C30CF8      JMP     PO
;
;READER: ;READER CHARACTER IN TO REG-A
;          (EXACTLY THE SAME AS MDS CALL)
3F04 C306F8      JMP     RI
;
;HOME: ;MOVE TO HOME POSITION
;          TREAT AS TRACK 00 SEEK
3F07 0E00        MVI     C,0
3F09 C32A3F      JMP     SETTRK
;
;SELDSK: ;SELECT DISK GIVEN BY REGISTER C
;          CP/M HAS CHECKED FOR DISK SELECT 0 - 3, BUT WE MAY HAVE
;          A SMALLER MDS SYSTEM, SO CHECK AGAIN AND GIVE ERROR
;          BY CALLING MON80
3F0C 79          MOV     A,C
3F0D FE04        CPI     NDISKS ;TOO LARGE?
3F0F D40FF8      CNC     RMON80 ;GIVES #ADDR MESSAGE AT CONSOLE
;
3F12 E602        ANI     10B     ;00 00 FOR DRIVE 0,1 AND 10 10 FOR DRIVE 2,3
3F14 32DF3F      STA     DBANK   ;TO SELECT DRIVE BANK
3F17 79          MOV     A,C     ;00, 01, 10, 11
3F18 E601        ANI     1B      ;MDS HAS 0,1 AT 78, 2,3 AT 88
3F1A B7          ORA     A       ;RESULT 00?
3F1B CA203F      JZ      SETDRIVE
3F1E 3E30        MVI     A,00110000B ;SELECTS DRIVE 1 IN BANK
SETDRIVE:
3F20 4F          MOV     C,A     ;SAVE THE FUNCTION
3F21 21E13F      LXI     H,IOF   ;IO FUNCTION
3F24 7E          MOV     A,M
3F25 E6CF        ANI     11001111B ;MASK OUT DISK NUMBER
3F27 B1          ORA     C       ;MASK IN NEW DISK NUMBER
3F28 77          MOV     M,A     ;SAVE IT IN IOPB
3F29 C9          RET
;
;
SETTRK: ;SET TRACK ADDRESS GIVEN BY C
3F2A 21E33F      LXI     H,IOT
3F2D 71          MOV     M,C

```

```

3F2E C9          RET
;
SETSEC: ;SET SECTOR NUMBER GIVEN BY C
3F2F 79          MOV     A,C      ;SECTOR NUMBER TO ACCUM
3F30 32E43F      STA     IOS      ;STORE SECTOR NUMBER TO IOPB
3F33 C9          RET
;
SETDMA: ;SET DMA ADDRESS GIVEN BY REGS B,C
3F34 69          MOV     L,C
3F35 60          MOV     H,B
3F36 22E53F      SHLD    IOD
3F39 C9          RET
;
READ:  ;READ NEXT DISK RECORD (ASSUMING DISK/TRK/SEC/DMA SET)
3F3A 0E04        MVI     C,READF ;SET TO READ FUNCTION
3F3C CD593F      CALL    SETFUNC
3F3F CD693F      CALL    WAITIO ;PERFORM READ FUNCTION
3F42 C9          RET             ;MAY HAVE ERROR SET IN REG-A
;
;
WRITE: ;DISK WRITE FUNCTION
3F43 0E06        MVI     C,WRITEF
3F45 CD593F      CALL    SETFUNC ;SET TO WRITE FUNCTION
3F48 CD693F      CALL    WAITIO
3F4B C9          RET             ;MAY HAVE ERROR SET
;
;
; UTILITY SUBROUTINES
PRMSG: ;PRINT MESSAGE AT H,L TO 0
3F4C 7E          MOV     A,M
3F4D B7          ORA     A        ;ZERO?
3F4E C8          RZ
;
; MORE TO PRINT
3F4F E5          PUSH    H
3F50 4F          MOV     C,A
3F51 CDFB3E      CALL    CONOUT
3F54 E1          POP     H
3F55 23          INX     H
3F56 C34C3F      JMP     PRMSG
;
SETFUNC:
; SET FUNCTION FOR NEXT I/O (COMMAND IN REG-C)
3F59 21E13F      LXI     H,IOF    ;IO FUNCTION ADDRESS
3F5C 7E          MOV     A,M      ;GET IT TO ACCUMULATOR FOR MASKING
3F5D E6F8        ANI     11111000B ;REMOVE PREVIOUS COMMAND
3F5F B1          ORA     C        ;SET TO NEW COMMAND
3F60 77          MOV     M,A      ;REPLACED IN IOPB
; THE MDS-800 CONTROLLER REQUIRES DISK BANK BIT IN SECTOR BYTE
; MASK THE BIT FROM THE CURRENT I/O FUNCTION
3F61 E620        ANI     00100000B ;MASK THE DISK SELECT BIT

```

3F63 21E43F	LXI	H,IOS	;ADDRESS THE SECTOR SELECT BYTE
3F66 B6	ORA	M	;SELECT PROPER DISK BANK
3F67 77	MOV	M,A	;SET DISK SELECT BIT ON/OFF
3F68 C9	RET		
;			
3F69 0E0A	MVI	C,RETRY	;MAX RETRIES BEFORE PERM ERROR
REWAIT:			
;			
3F6B CDB83F	CALL	INTYPE	;IN RTYPE
3F6E CDC53F	CALL	INBYTE	;CLEARS THE CONTROLLER
;			
3F71 3ADF3F	LDA	DBANK	;SET BANK FLAGS
3F74 B7	ORA	A	;ZERO IF DRIVE 0,1 AND NZ IF 2,3
3F75 3EE0	MVI	A,IOPB AND 0FFH	;LOW ADDRESS FOR IOPB
3F77 063F	MVI	B,IOPB SHR 8	;HIGH ADDRESS FOR IOPB
3F79 C2843F	JNZ	IODR1	;DRIVE BANK 1?
3F7C D379	OUT	ILOW	;LOW ADDRESS TO CONTROLLER
3F7E 78	MOV	A,B	
3F7F D37A	OUT	IHIGH	;HIGH ADDRESS
3F81 C3893F	JMP	WAIT0	;TO WAIT FOR COMPLETE
;			
3F84 D389	OUT	ILOW+10H	;88 FOR DRIVE BANK 10
3F86 78	MOV	A,B	
3F87 D38A	OUT	IHIGH+10H	
;			
3F89 CDD23F	CALL	INSTAT	;WAIT FOR COMPLETION
3F8C E604	ANI	IORDY	;READY?
3F8E CA893F	JZ	WAIT0	
;			
CHECK IO COMPLETION OK			
3F91 CDB83F	CALL	INTYPE	;MUST BE IO COMPLETE (00) UNLINKED
00 UNLINKED I/O COMPLETE, 01 LINKED I/O COMPLETE (NOT USED)			
10 DISK STATUS CHANGED 11 (NOT USED)			
3F94 FE02	CPI	10B	;READY STATUS CHANGE?
3F96 CAAB3F	JZ	WREADY	
;			
MUST BE 00 IN THE ACCUMULATOR			
3F99 B7	ORA	A	
3F9A C2B13F	JNZ	WERROR	;SOME OTHER CONDITION, RETRY
;			
CHECK I/O ERROR BITS			
3F9D CDC53F	CALL	INBYTE	
3FA0 17	RAL		
3FA1 DAAB3F	JC	WREADY	;UNIT NOT READY
3FA4 1F	RAR		
3FA5 E6FE	ANI	11111110B	;ANY OTHER ERRORS? (DELETED DATA OK)
3FA7 C2B13F	JNZ	WERROR	
;			

```

;      READ OR WRITE IS OK, ACCUMULATOR CONTAINS ZERO
3FAA C9      RET
;
WREADY: ;NOT READY, TREAT AS ERROR FOR NOW
3FAB CDC53F  CALL    INBYTE      ;CLEAR RESULT BYTE
3FAE C3B13F  JMP      TRYCOUNT
;
WERROR: ;RETURN HARDWARE MALFUNCTION (CRC, TRACK, SEEK, ETC.)
;      THE MDS CONTROLLER HAS RETURNED A BIT IN EACH POSITION
;      OF THE ACCUMULATOR, CORRESPONDING TO THE CONDITIONS:
;      0      - DELETED DATA (ACCEPTED AS OK ABOVE)
;      1      - CRC ERROR
;      2      - SEEK ERROR
;      3      - ADDRESS ERROR (HARDWARE MALFUNCTION)
;      4      - DATA OVER/UNDER FLOW (HARDWARE MALFUNCTION)
;      5      - WRITE PROTECT (TREATED AS NOT READY)
;      6      - WRITE ERROR (HARDWARE MALFUNCTION)
;      7      - NOT READY
;      (ACCUMULATOR BITS ARE NUMBERED 7 6 5 4 3 2 1 0)
;
;      IT MAY BE USEFUL TO FILTER OUT THE VARIOUS CONDITIONS,
;      BUT WE WILL GET A PERMANENT ERROR MESSAGE IF IT IS NOT
;      RECOVERABLE. IN ANY CASE, THE NOT READY CONDITION IS
;      TREATED AS A SEPARATE CONDITION FOR LATER IMPROVEMENT
TRYCOUNT:
;      REGISTER C CONTAINS RETRY COUNT, DECREMENT 'TIL ZERO
3FB1 0D      DCR      C
3FE2 C26B3F  JNZ      REWAIT ;FOR ANOTHER TRY
;
;      CANNOT RECOVER FROM ERROR
3FB5 3E01    MVI      A,1      ;ERROR CODE
3FB7 C9      RET
;
;      INTYPE, INBYTE, INSTAT READ DRIVE BANK 00 OR 10
3FB8 3ADF3F  INTYPE: LDA    DBANK
3FBB B7      ORA      A
3FEC C2C23F  JNZ      INTYP1 ;SKIP TO BANK 10
3FBF DB79    IN      RTYPE
3FC1 C9      RET
3FC2 DB89    INTYP1: IN     RTYPE+10H      ;78 FOR 0,1 88 FOR 2,3
3FC4 C9      RET
;
3FC5 3ADF3F  INBYTE: LDA    DBANK
3FC8 B7      ORA      A
3FC9 C2CF3F  JNZ      INBYT1
3FCC DB7B    IN      RBYTE
3FCE C9      RET
3FCF DB8B    INBYT1: IN     RBYTE+10H
3FD1 C9      RET
;

```

```

3FD2 3ADF3F  INSTAT: LCA    DBANK
3FD5 B7      ORA      A
3FD6 C2DC3F  JNZ      INSTAL
3FD9 DB78    IN       DSTAT
3FDB C9      RET
3FDC DB88    INSTAL: IN    DSTAT+10H
3FDE C9      RET
;
;
;
;      DATA AREAS (MUST BE IN RAM)
3FDF 00      DBANK: DB     0      ;DISK BANK 00 IF DRIVE 0,1
;                      ;      10 IF DRIVE 2,3
IOPB: ;IO PARAMETER BLOCK
3FE0 80      DB      80H      ;NORMAL I/O OPERATION
3FE1 04      IOF:   DB      READF ;IO FUNCTION, INITIAL READ
3FE2 01      ION:   DB      1    ;NUMBER OF SECTORS TO READ
3FE3 02      IOT:   DB      OFFSET ;TRACK NUMBER
3FE4 01      IOS:   DB      1    ;SECTOR NUMBER
3FE5 8000     IOD:   DW      BUFF ;IO ADDRESS
;
;
3FE7          END

```

APPENDIX D: A SKELETAL CBIOS

```

; SKELETAL CBIOS FOR FIRST LEVEL OF CP/M ALTERATION
;
; NOTE : MSIZE DETERMINES WHERE THIS CBIOS IS LOCATED
0010 = MSIZE EQU 16 ;CP/M VERSION MEMORY SIZE IN KILOBYTES
3E00 = PATCH EQU MSIZE*1024-2*256 ;START OF THE CBIOS PATCH
;
; WE WILL USE THE AREA RESERVED STARTING AT LOCATION
; 40H IN PAGE 0 FOR HOLDING THE VALUES OF:
; TRACK = LAST SELECTED TRACK
; SECTOR = LAST SELECTED SECTOR
; DMAAD = LAST SELECTED DMA ADDRESS
; DISKNO = LAST SELECTED DISK NUMBER
; (NOTE THAT ALL ARE BYTE VALUES EXCEPT FOR DMAAD)
;
0040 = SCRAT EQU 40H ;BASE OF SCRATCH AREA (FROM 40H TO 4FH)
0040 = TRACK EQU SCRAT ;CURRENTLY SELECTED TRACK
0041 = SECTOR EQU SCRAT+1 ;CURRENTLY SELECTED SECTOR
0042 = DMAAD EQU SCRAT+2 ;CURRENT DMA ADDRESS
0044 = DISKNO EQU SCRAT+4 ;CURRENT DISK NUMBER
;
;
3E00 ORG PATCH ;ORIGIN OF THIS PROGRAM
0000 = CBASE EQU (MSIZE-16)*1024 ;BIAS FOR SYSTEMS LARGER THAN 16K
2900 = CPMB EQU CBASE+2900H ;BASE OF CP/M (= BASE OF CCP)
3106 = BDOS EQU CBASE+3106H ;BASE OF RESIDENT PORTION OF CP/M
1500 = CPML EQU $-CPMB ;LENGTH OF THE CP/M SYSTEM IN BYTES
002A = NSECTS EQU CPML/128 ;NUMBER OF SECTORS TO LOAD ON WARM START
;
; JUMP VECTOR FOR INDIVIDUAL SUBROUTINES
3E00 C32D3E JMP BOOT ;COLD START
WBOOTE:
3E03 C3303E JMP WBOOT ;WARM START
3E06 C3993E JMP CONST ;CONSOLE STATUS
3E09 C3AC3E JMP CONIN ;CONSOLE CHARACTER IN
3E0C C3BF3E JMP CONOUT ;CONSOLE CHARACTER OUT
3E0F C3D13E JMP LIST ;LIST CHARACTER OUT
3E12 C3D33E JMP PUNCH ;PUNCH CHARACTER OUT
3E15 C3D53E JMP READER ;READER CHARACTER OUT
3E18 C3DA3E JMP HOME ;MOVE HEAD TO HOME POSITION
3E1B C3E03E JMP SELDSK ;SELECT DISK
3E1E C3F53E JMP SETTRK ;SET TRACK NUMBER
3E21 C30A3F JMP SETSEC ;SET SECTOR NUMBER
3E24 C31F3F JMP SETDMA ;SET DMA ADDRESS
3E27 C3353F JMP READ ;READ DISK
3E2A C3483F JMP WRITE ;WRITE DISK
;
;

```

```

; INDIVIDUAL SUBROUTINES TO PERFORM EACH FUNCTION
BOOT: ;SIMPLEST CASE IS TO JUST PERFORM PARAMETER INITIALIZATION
3E2D C3793E JMP GOCPM ;INITIALIZE AND GO TO CP/M

;
WBOOT: ;SIMPLEST CASE IS TO READ THE DISK UNTIL ALL SECTORS LOADED
3E30 318000 LXI SP,80H ;USE SPACE BELOW BUFFER FOR STACK
3E33 0E00 MVI C,0 ;SELECT DISK 0
3E35 CDE03E CALL SELDSK
3E38 CDDA3E CALL HOME ;GO TO TRACK 00

;
3E3B 062A MVI B,NSECTS ;B COUNTS THE NUMBER OF SECTORS TO LOAD
3E3D 0E00 MVI C,0 ;C HAS THE CURRENT TRACK NUMBER
3E3F 1602 MVI D,2 ;D HAS THE NEXT SECTOR TO READ

; NOTE THAT WE BEGIN BY READING TRACK 0, SECTOR 2 SINCE SECTOR 1
; CONTAINS THE COLD START LOADER, WHICH IS SKIPPED IN A WARM START
3E41 210029 LXI H,CPMB ;BASE OF CP/M (INITIAL LOAD POINT)

LOAD1: ;LOAD ONE MORE SECTOR
3E44 C5 PUSH B ;SAVE SECTOR COUNT, CURRENT TRACK
3E45 D5 PUSH D ;SAVE NEXT SECTOR TO READ
3E46 E5 PUSH H ;SAVE DMA ADDRESS
3E47 4A MOV C,D ;GET SECTOR ADDRESS TO REGISTER C
3E48 CD0A3F CALL SETSEC ;SET SECTOR ADDRESS FROM REGISTER C
3E4B C1 POP B ;RECALL DMA ADDRESS TO B,C
3E4C C5 PUSH B ;REPLACE ON STACK FOR LATER RECALL
3E4D CD1F3F CALL SETDMA ;SET DMA ADDRESS FROM B,C

;
; DRIVE SET TO 0, TRACK SET, SECTOR SET, DMA ADDRESS SET
3E50 CD353F CALL READ
3E53 FE00 CPI 00H ;ANY ERRORS?
3E55 C2303E JNZ WBOOT ;RETRY THE ENTIRE BOOT IF AN ERROR OCCURS

;
; NO ERROR, MOVE TO NEXT SECTOR
3E58 E1 POP H ;RECALL DMA ADDRESS
3E59 118000 LXI D,128 ;DMA=DMA+128
3E5C 19 DAD D ;NEW DMA ADDRESS IS IN H,L
3E5D D1 POP D ;RECALL SECTOR ADDRESS
3E5E C1 POP B ;RECALL NUMBER OF SECTORS REMAINING, AND CURRENT TRK
3E5F 05 DCR B ;SECTORS=SECTORS-1
3E60 CA793E JZ GOCPM ;TRANSFER TO CP/M IF ALL HAVE BEEN LOADED

;
; MORE SECTORS REMAIN TO LOAD, CHECK FOR TRACK CHANGE
3E63 14 INR D
3E64 7A MOV A,D ;SECTOR=27?, IF SO, CHANGE TRACKS
3E65 FE1B CPI 27
3E67 DA443E JC LOAD1 ;CARRY GENERATED IF SECTOR<27

;
; END OF CURRENT TRACK, GO TO NEXT TRACK
3E6A 1601 MVI D,1 ;BEGIN WITH FIRST SECTOR OF NEXT TRACK
3E6C 0C INR C ;TRACK=TRACK+1

```

```

;      SAVE REGISTER STATE, AND CHANGE TRACKS
3E6D C5      PUSH      B
3E6E D5      PUSH      D
3E6F E5      PUSH      H
3E70 CDF53E  CALL      SETTRK ;TRACK ADDRESS SET FROM REGISTER C
3E73 E1      POP       H
3E74 D1      POP       D
3E75 C1      POP       B
3E76 C3443E  JMP       LOAD1 ;FOR ANOTHER SECTOR

;
;      END OF LOAD OPERATION, SET PARAMETERS AND GO TO CP/M
GOCPM:
3E79 3EC3    MVI       A,0C3H ;C3 IS A JMP INSTRUCTION
3E7B 320000  STA       0 ;FOR JMP TO WBOOT
3E7E 21033E  LXI       H,WBOOTE ;WBOOT ENTRY POINT
3E81 220100  SHLD      1 ;SET ADDRESS FIELD FOR JMP AT 0

;
3E84 320500  STA       5 ;FOR JMP TO BDOS
3E87 210631  LXI       H,BDOS ;BDOS ENTRY POINT
3E8A 220600  SHLD      6 ;ADDRESS FIELD OF JUMP AT 5 TO BDOS

;
3E8D 018000  LXI       B,80H ;DEFAULT DMA ADDRESS IS 80H
3E90 CD1F3F  CALL      SETDMA

;
3E93 FB      EI ;ENABLE THE INTERRUPT SYSTEM
;      FUTURE VERSIONS OF CCP WILL SELECT THE DISK GIVEN BY REGISTER
;      C UPON ENTRY, HENCE ZERO IT IN THIS VERSION OF THE BIOS FOR
;      FUTURE COMPATIBILITY.
3E94 0E00    MVI       C,0 ;SELECT DISK ZERO AFTER INITIALIZATION
3E96 C30029  JMP       CPMB ;GO TO CP/M FOR FURTHER PROCESSING

;
;
;      SIMPLE I/O HANDLERS (MUST BE FILLED IN BY USER)
;      IN EACH CASE, THE ENTRY POINT IS PROVIDED, WITH SPACE RESERVED
;      TO INSERT YOUR OWN CODE
;
CONST: ;CONSOLE STATUS, RETURN 0FFH IF CHARACTER READY, 00H IF NOT
3E99 DS      10H ;SPACE FOR STATUS SUBROUTINE
3EA9 3E00    MVI       A,00H
3EAB C9      RET

;
CONIN: ;CONSOLE CHARACTER INTO REGISTER A
3EAC DS      10H ;SPACE FOR INPUT ROUTINE
3EBC E67F    ANI       7FH ;STRIP PARITY BIT
3EBE C9      RET

;
CONOUT: ;CONSOLE CHARACTER OUTPUT FROM REGISTER C
3EBF 79      MOV       A,C ;GET TO ACCUMULATOR
3EC0 DS      10H ;SPACE FOR OUTPUT ROUTINE
3ED0 C9      RET

```

```

;
LIST: ;LIST CHARACTER FROM REGISTER C
3ED1 79      MOV      A,C      ;CHARACTER TO REGISTER A
3ED2 C9      RET              ;NULL SUBROUTINE

;
PUNCH: ;PUNCH CHARACTER FROM REGISTER C
3ED3 79      MOV      A,C      ;CHARACTER TO REGISTER A
3ED4 C9      RET              ;NULL SUBROUTINE

;
;
READER: ;READ CHARACTER INTO REGISTER A FROM READER DEVICE
3ED5 3E1A    MVI      A,1AH    ;ENTER END OF FILE FOR NOW (REPLACE LATER)
3ED7 E67F    ANI      7FH      ;REMEMBER TO STRIP PARITY BIT
3ED9 C9      RET

;
;
; I/O DRIVERS FOR THE DISK FOLLOW
; FOR NOW, WE WILL SIMPLY STORE THE PARAMETERS AWAY FOR USE
; IN THE READ AND WRITE SUBROUTINES
;
HOME: ;MOVE TO THE TRACK 00 POSITION OF CURRENT DRIVE
; TRANSLATE THIS CALL INTO A SETTRK CALL WITH PARAMETER 00
3EDA 0E00    MVI      C,0      ;SELECT TRACK 0
3EDC CDF53E  CALL     SETTRK
3EDF C9      RET              ;WE WILL MOVE TO 00 ON FIRST READ/WRITE

;
SELDSK: ;SELECT DISK GIVEN BY REGISTER C
3EE0 79      MOV      A,C
3EE1 324400  STA      DISKNO
3EE4         DS       10H      ;SPACE FOR DISK SELECTION ROUTINE
3EF4 C9      RET

;
SETTRK: ;SET TRACK GIVEN BY REGISTER C
3EF5 79      MOV      A,C
3EF6 324000  STA      TRACK
3EF9         DS       10H      ;SPACE FOR TRACK SELECT
3F09 C9      RET

;
SETSEC: ;SET SECTOR GIVEN BY REGISTER C
3F0A 79      MOV      A,C
3F0B 324100  STA      SECTOR
3F0E         DS       10H      ;SPACE FOR SECTOR SELECT
3F1E C9      RET

;
SETDMA: ;SET DMA ADDRESS GIVEN BY REGISTERS B AND C
3F1F 69      MOV      L,C      ;LOW ORDER ADDRESS
3F20 60      MOV      H,B      ;HIGH ORDER ADDRESS
3F21 224200  SHLD     DMAAD    ;SAVE THE ADDRESS
3F24         DS       10H      ;SPACE FOR SETTING THE DMA ADDRESS
3F34 C9      RET

```

```

;
READ:  ;PERFORM READ OPERATION (USUALLY THIS IS SIMILAR TO WRITE
;      SO WE WILL ALLOW SPACE TO SET UP READ COMMAND, THEN USE
;      COMMON CODE IN WRITE)
3F35      DS      10H      ;SET UP READ COMMAND
3F45 C3583F  JMP      WAITIO ;TO PERFORM THE ACTUAL I/O
;
WRITE:  ;PERFORM A WRITE OPERATION
3F48      DS      10H      ;SET UP 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, WE HAVE SAVED THE DISK NUMBER IN 'DISKNO' (0,1)
;      THE TRACK NUMBER IN 'TRACK' (0-76)
;      THE SECTOR NUMBER IN 'SECTOR' (1-26)
;      THE DMA ADDRESS IN 'DMAAD' (0-65535)
;      ALL REMAINING SPACE FROM $ THROUGH MSIZE*1024-1 IS AVAILABLE:
00A7 = LEFT EQU      (MSIZE*1024-1)-$      ;SPACE REMAINING IN CBIOS
;
3F58 3E01      MVI      A,1      ;ERROR CONDITION
3F5A C9      RET      ;REPLACED WHEN FILLED-IN
3F5B      END

```

APPENDIX E: A SKELETAL GETSYS/PUTSYS PROGRAM

```

;      COMBINED GETSYS AND PUTSYS PROGRAMS FROM SECTION 4
;
;      START THE PROGRAMS AT THE BASE OF THE TRANSIENT PROGRAM AREA
0100      ORG      100H
0010 =    MSIZE    EQU      16      ;SIZE OF MEMORY IN KILOBYTES
;      BIAS IS THE AMOUNT TO ADD TO ADDRESSES FOR SYSTEMS LARGER THAN 16K
;      (REFERRED TO AS 'B' THROUGHOUT THE TEXT)
0000 =    BIAS     EQU      (MSIZE-16)*1024
;
;      GETSYS PROGRAM - READ TRACKS 0 AND 1 TO MEMORY AT 2880H+BIAS
;      REGISTER      USE
;      A              (SCRATCH REGISTER)
;      B              TRACK COUNT (0...76)
;      C              SECTOR COUNT (1...26)
;      D,E            (SCRATCH REGISTER PAIR)
;      H,L            LOAD ADDRESS
;      SP             SET TO STACK ADDRESS
;
GSTART:
0100 318028      LXI      SP,2880H+BIAS      ;START OF THE GETSYS PROGRAM
0103 218028      LXI      H,2880H+BIAS      ;SET STACK POINTER TO SCRATCH AREA
0106 0600        MVI      B,0                ;SET BASE LOAD ADDRESS
;
RDTRK:           MVI      C,1                ;START WITH TRACK 00
;
RDSEC:           ;READ FIRST (NEXT) TRACK
;
;      READ STARTING WITH SECTOR 1
0108 0E01        MVI      C,1
;
;      READ NEXT SECTOR
010A CD0003      CALL     READSEC
;
010D 118000      LXI      D,128              ;CHANGE LOAD ADDRESS TO NEXT 1/2 PAGE
0110 19          DAD      D                  ;HL=HL+128 TO NEXT ADDRESS
0111 0C          INR      C                  ;SECTOR=SECTOR+1
0112 79          MOV      A,C                ;CHECK FOR END OF TRACK
0113 FE1B        CPI      27
0115 DA0A01      JC       RDSEC              ;CARRY GENERATED IF C<27
;
;      ARRIVE HERE AT END OF TRACK, MOVE TO NEXT TRACK
0118 04          INR      B                  ;TRACK=TRACK+1
0119 78          MOV      A,B                ;CHECK FOR LAST TRACK
011A FE02        CPI      2                  ;TRACK=2?
011C DA0801      JC       RDTRK              ;CARRY GENERATED IF TRACK < 2
;
;      ARRIVE HERE AT END OF LOAD, HALT FOR NOW
011F FB          EI
0120 76          HLT
;
;      PUTSYS PROGRAM - PLACE MEMORY STARTING AT 2880H+BIAS BACK TO TRACKS
;      0 AND 1. START THIS PROGRAM ON THE NEXT PAGE
0200      ORG      ($+100H) AND 0FF00H

```

```

; REGISTER USE
; A (SCRATCH REGISTER)
; B TRACK COUNT (0,1)
; C SECTOR COUNT (1...26)
; D,E (SCRATCH REGISTER PAIR)
; H,L DUMP ADDRESS
; SP SET TO STACK ADDRESS
;
PSTART: ;START OF THE PUTSYS PROGRAM
0200 318028 LXI SP,2880H+BIAS ;SET STACK POINTER TO SCRATCH AREA
0203 218028 LXI H,2880H+BIAS ;SET BASE DUMP ADDRESS
0206 0600 MVI B,0 ;START WITH TRACK 0
WRTRK: ;WRITE FIRST (NEXT) TRACK
0208 0E01 MVI C,1 ;START WRITING AT SECTOR 1
WRSEC: ;WRITE FIRST (NEXT) SECTOR
020A CD8003 CALL WRITESEC ;PERFORM THE WRITE
020D 118000 LXI D,128 ;MOVE DUMP ADDRESS TO NEXT 1/2 PAGE
0210 19 DAD D ;HL=HL+128
0211 0C INR C ;SECTOR=SECTOR+1
0212 79 MOV A,C ;CHECK FOR END OF TRACK
0213 FE1B CPI 27 ;SECTOR=27?
0215 DA0A02 JC WRSEC ;CARRY GENERATED IF SECTOR < 27
;
; ARRIVE HERE AT END OF TRACK, MOVE TO NEXT TRACK
0218 04 INR B ;TRACK=TRACK+1
0219 78 MOV A,B ;TEST FOR LAST TRACK
021A FE02 CPI 2 ;TRACK=2?
021C DA0802 JC WRTRK ;CARRY GENERATED IF TRACK < 2
;
; ARRIVE HERE AT END OF DUMP, HALT FOR NOW
021F FB EI
0220 76 HLT
;
;
; USER-SUPPLIED SUBROUTINES FOR SECTOR READ AND SECTOR WRITE
;
; MOVE TO NEXT PAGE FOR READSEC AND WRITESEC
0300 ORG ($+100H) AND 0FF00H
;
READSEC: ;READ THE NEXT SECTOR
; TRACK TO READ IS IN REGISTER B
; SECTOR TO READ IS IN REGISTER C
; BRANCH TO LABEL GSTART IF ERROR OCCURS
; READ 128 BYTES OF DATA TO ADDRESS GIVEN BY H,L
0300 C5 PUSH B
0301 E5 PUSH H
; ** PLACE READ OPERATION HERE **
0302 E1 POP H
0303 C1 POP B
0304 C9 RET

```

```

;
;      MOVE TO NEXT 1/2 PAGE FOR WRITESEC SUBROUTINE
0380      ORG      ($ AND 0FF00H) + 80H
WRITESEC:      ;WRITE THE NEXT SECTOR
;      TRACK TO WRITE IS IN REGISTER B
;      SECTOR TO WRITE IS IN REGISTER C
;      BRANCH TO LABEL PSTART IF ERROR OCCURS
;      WRITE 128 BYTES OF DATA FROM ADDRESS GIVEN BY H,L
0380 C5      PUSH    B
0381 E5      PUSH    H
;      ** PLACE WRITE OPERATION HERE **
0382 E1      POP     H
0383 C1      POP     B
0384 C9      RET
;
;      END OF GETSYS/PUTSYS PROGRAM
0385      END

```

APPENDIX F: A SKELETAL COLD START LOADER

```

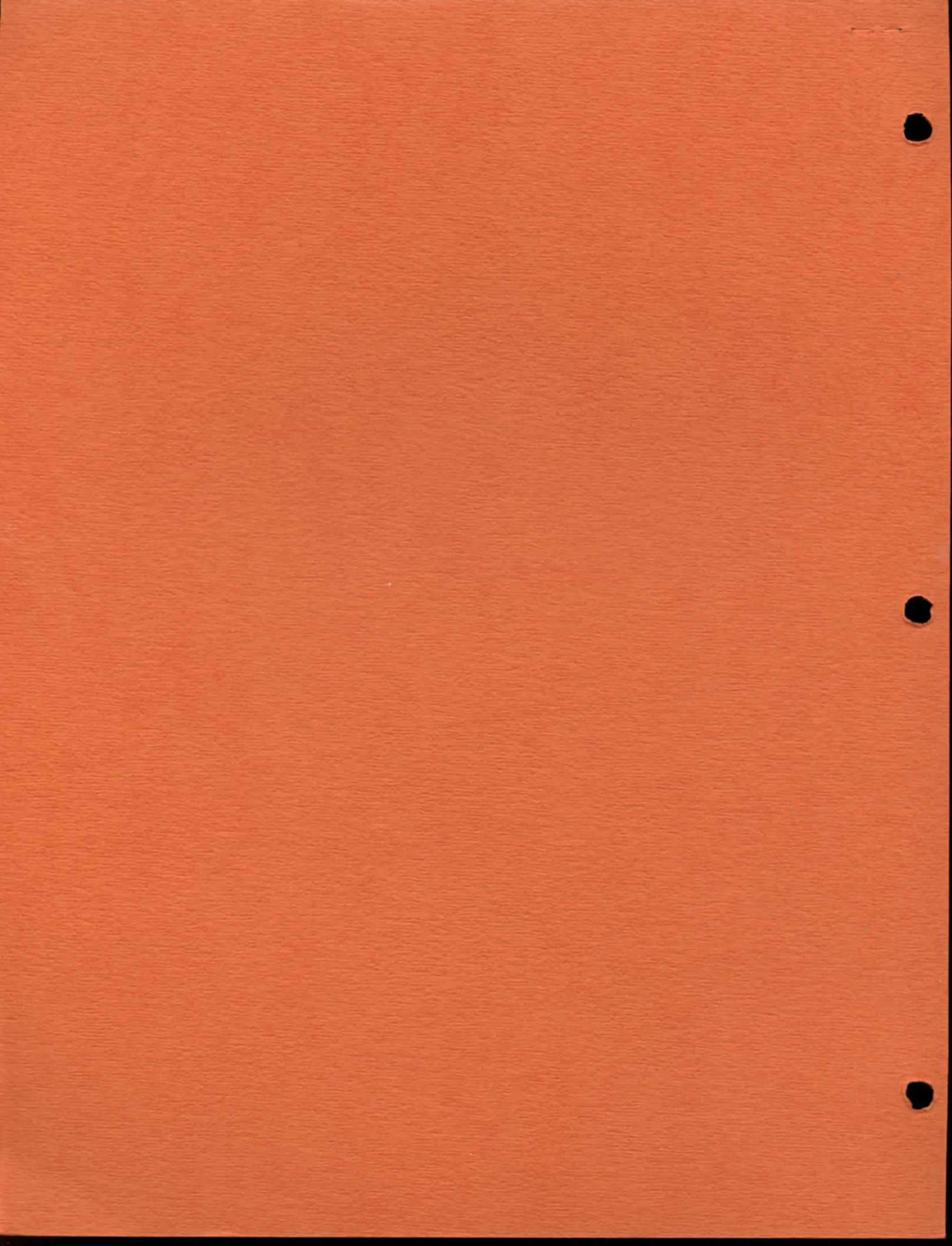
; THIS IS A SAMPLE COLD START LOADER WHICH, WHEN MODIFIED, RESIDES
; ON TRACK 00, SECTOR 01 (THE FIRST SECTOR ON THE DISKETTE). WE
; ASSUME THAT THE CONTROLLER HAS LOADED THIS SECTOR INTO MEMORY
; UPON SYSTEM STARTUP (THIS PROGRAM CAN BE KEYED-IN, OR EXIST IN
; A PAGE OF READ-ONLY MEMORY BEYOND THE ADDRESS SPACE OF THE CP/M
; VERSION YOU ARE RUNNING). THE COLD START LOADER BRINGS THE CP/M
; SYSTEM INTO MEMORY AT 'LOADP' (NOMINALLY 2900H) + 'BIAS' WHERE
; THE BIAS VALUE ACCOUNTS FOR MEMORY SYSTEMS LARGER THAN 16K, AND
; CP/M VERSIONS WHICH HANDLE THE LARGER MEMORY SPACE. IN A 16K
; SYSTEM, THE VALUE OF BIAS IS 0000H. AFTER LOADING THE CP/M SYS-
; TEM, THE COLD START LOADER BRANCHES TO THE 'BOOT' ENTRY POINT OF
; THE BIOS, WHICH BEGINS AT 'BIOS' + 'BIAS'. THE COLD START LOADER
; IS NOT USED AGAIN UNTIL THE SYSTEM IS POWERED UP AGAIN, AS LONG
; AS THE BIOS IS NOT OVERWRITTEN.
;
; THE ORIGIN IS 0, ASSUMING THE CONTROLLER LOADS THE COLD START
; PROGRAM AT THE BASE OF MEMORY. THIS ORIGIN MUST BE IN HIGH
; MEMORY (BEYOND THE END OF THE BIOS) IF THE COLD START LOADER
; IS IMPLEMENTED IN READ-ONLY-MEMORY.
0000 ORG 0000H ;BASE OF MEMORY
0010 = MSIZE EQU 16 ;MEMORY SIZE IN KILOBYTES
0020 = BIAS EQU (MSIZE-16)*1024 ;BIAS TO ADD TO LOAD ADDRESSES
2900 = LOADP EQU 2900H ;LOAD POINT FOR CP/M SYSTEM
3E00 = BIOS EQU 3E00H ;BASIC I/O SYSTEM (2 PAGES = 512 BYTES)
3E00 = BOOT EQU BIOS ;COLD START ENTRY POINT IN BIOS
1700 = SIZE EQU BIOS+512-LOADP ;SIZE OF THE CP/M SYSTEM TO LOAD
002E = SECTS EQU SIZE/128 ;NUMBER OF SECTORS TO LOAD
;
; BEGIN THE LOAD OPERATION
0000 010200 COLD: LXI B,2 ;CLEAR B TO 0, SET C TO SECTOR 2
0003 162E MVI D,SECTS ;NUMBER OF SECTORS TO LOAD IS IN D
0005 210229 LXI H,LOADP+BIAS ;LOAD POINT IN H,L
;
LSECT: ;LOAD NEXT SECTOR
; INSERT INLINE CODE AT THIS POINT TO READ ONE 128 BYTE SECTOR
; FROM TRACK GIVEN BY REGISTER B,
; SECTOR GIVEN BY REGISTER C,
; INTO ADDRESS GIVEN BY REGISTER PAIR H,L
; BRANCH TO LOCATION 'COLD' IF A READ ERROR OCCURS
;
; *****
; USER SUPPLIED READ OPERATION GOES HERE
; *****
; (SPACE IS RESERVED FOR YOUR PATCH)
0008 C36B00 JMP PASTPATCH ;REMOVE THIS JUMP WHEN PATCHED
000B DS 60H

```

```

;
PASTPATCH:
;      GO TO NEXT SECTOR IF LOAD IS INCOMPLETE
006B 15      DCR      D      ;SECTS=SECTS-1
006C CA003E  JZ       BOOT+BIAS      ;GO TO BOOT LOADER AT 3E00H+BIAS
;
;      MORE SECTORS TO LOAD
;      USE SP FOR SCRATCH REGISTER TO HOLD LOAD ADDRESS INCREMENT
006F 318000  LXI      SP,128
0072 39      DAD      SP      ;HL=HL+128 TO NEXT LOAD ADDRESS
;
;      INR      C      ;SECTOR=SECTOR+1
0073 0C      MOV      A,C      ;MOVE SECTOR COUNT TO A FOR COMPARE
0074 79      CPI      27      ;END OF CURRENT TRACK?
0075 FE1B    JC       LSECT     ;CARRY GENERATED IF SECTOR < 27
0077 DA0800
;
;      END OF TRACK, MOVE TO NEXT TRACK
007A 0E01    MVI      C,1      ;SECTOR=1
007C 04      INR      B      ;TRACK=TRACK+1
007D C30800  JMP      LSECT     ;FOR ANOTHER SECTOR
;
0080      END

```



II DIGITAL RESEARCH

Post Office Box 579, Pacific Grove, California 93950, (408) 649-3896

ED: A CONTEXT EDITOR FOR THE CP/M DISK SYSTEM USER'S MANUAL

COPYRIGHT (c) 1976, 1978

DIGITAL RESEARCH

Table of Contents

1.	ED TUTORIAL	1
1.1	Introduction to ED	1
1.2	ED Operation	1
1.3	Text Transfer Functions	1
1.4	Memory Buffer Organization	5
1.5	Memory Buffer Operation	5
1.6	Command Strings	7
1.7	Text Search and Alteration	8
1.8	Source Libraries	11
1.9	Repetitive Command Execution	12
2.	ED ERROR CONDITIONS	13
3.	CONTROL CHARACTERS AND COMMANDS	14

ED USER'S MANUAL

1. ED TUTORIAL

1.1. Introduction to ED.

ED is the context editor for CP/M, and is used to create and alter CP/M source files. ED is initiated in CP/M by typing

$$\text{ED} \left\{ \begin{array}{l} \langle \text{filename} \rangle \\ \langle \text{filename} \rangle . \langle \text{filetype} \rangle \end{array} \right\}$$

In general, ED reads segments of the source file given by `<filename>` or `<filename> . <filetype>` into central memory, where the file is manipulated by the operator, and subsequently written back to disk after alterations. If the source file does not exist before editing, it is created by ED and initialized to empty. The overall operation of ED is shown in Figure 1.

1.2. ED Operation

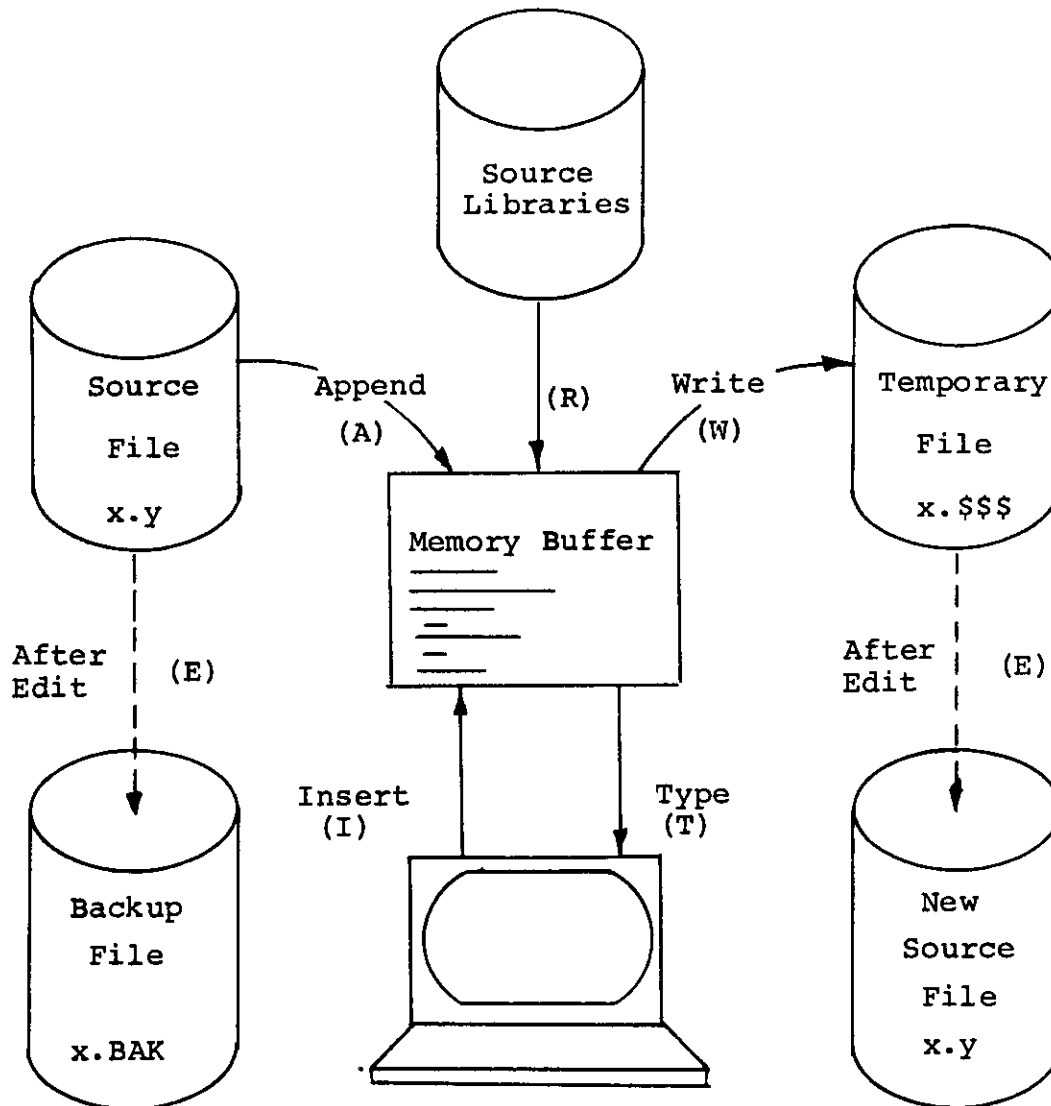
ED operates upon the source file, denoted in Figure 1 by `x.y`, and passes all text through a memory buffer where the text can be viewed or altered (the number of lines which can be maintained in the memory buffer varies with the line length, but has a total capacity of about 6000 characters in a 16K CP/M system). Text material which has been edited is written onto a temporary work file under command of the operator. Upon termination of the edit, the memory buffer is written to the temporary file, followed by any remaining (unread) text in the source file. The name of the original file is changed from `x.y` to `x.BAK` so that the most recent previously edited source file can be reclaimed if necessary (see the CP/M commands `ERASE` and `RENAME`). The temporary file is then changed from `x.$$$` to `x.y` which becomes the resulting edited file.

The memory buffer is logically between the source file and working file as shown in Figure 2.

1.3. Text Transfer Functions

Given that `n` is an integer value in the range 0 through 65535, the following ED commands transfer lines of text from the source file through the memory buffer to the temporary (and eventually final) file:

Figure 1. Overall ED Operation



Note: the ED program accepts both lower and upper case ASCII characters as input from the console. Single letter commands can be typed in either case. The U command can be issued to cause ED to translate lower case alphabets to upper case as characters are filled to the memory buffer from the console. Characters are echoed as typed without translation, however. The -U command causes ED to revert to "no translation" mode. ED starts with an assumed -U in effect.

Figure 2. Memory Buffer Organization

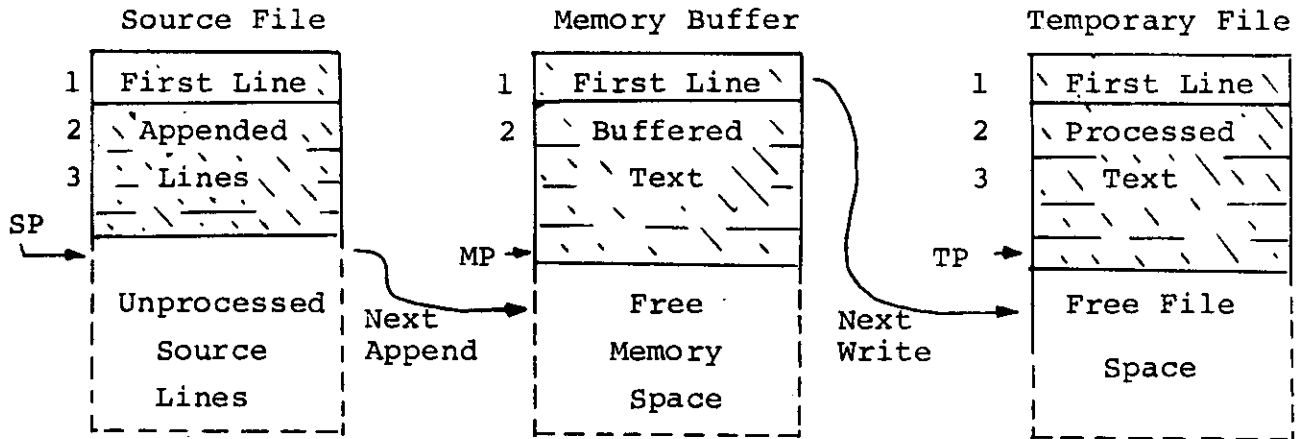
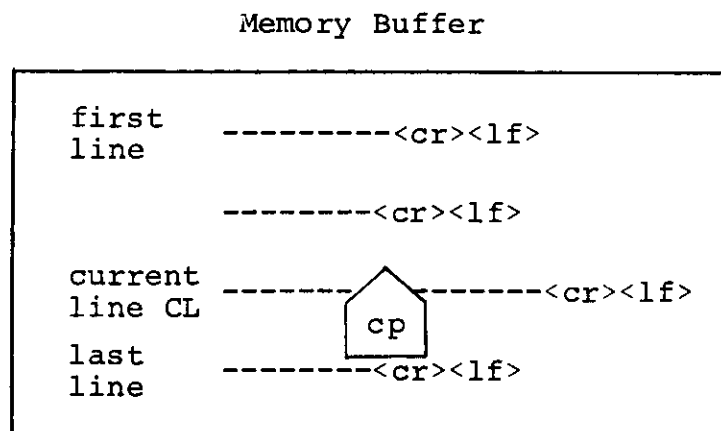


Figure 3. Logical Organization of Memory Buffer



- nA<cr>* - append the next n unprocessed source lines from the source file at SP to the end of the memory buffer at MP. Increment SP and MP by n.

- nW<cr> - write the first n lines of the memory buffer to the temporary file free space. Shift the remaining lines n+1 through MP to the top of the memory buffer. Increment TP by n.

- E<cr> - end the edit. Copy all buffered text to temporary file, and copy all unprocessed source lines to the temporary file. Rename files as described previously.

- H<cr> - move to head of new file by performing automatic E command. Temporary file becomes the new source file, the memory buffer is emptied, and a new temporary file is created (equivalent to issuing an E command, followed by a reinvocation of ED using x.y as the file to edit).

- O<cr> - return to original file. The memory buffer is emptied, the temporary file is deleted, and the SP is returned to position 1 of the source file. The effects of the previous editing commands are thus nullified.

- Q<cr> - quit edit with no file alterations, return to CP/M.

There are a number of special cases to consider. If the integer n is omitted in any ED command where an integer is allowed, then 1 is assumed. Thus, the commands A and W append one line and write 1 line, respectively. In addition, if a pound sign (#) is given in the place of n, then the integer 65535 is assumed (the largest value for n which is allowed). Since most reasonably sized source files can be contained entirely in the memory buffer, the command #A is often issued at the beginning of the edit to read the entire source file to memory. Similarly, the command #W writes the entire buffer to the temporary file. Two special forms of the A and W

*<cr> represents the carriage-return key

commands are provided as a convenience. The command 0A fills the current memory buffer to at least half-full, while 0W writes lines until the buffer is at least half empty. It should also be noted that an error is issued if the memory buffer size is exceeded. The operator may then enter any command (such as W) which does not increase memory requirements. The remainder of any partial line read during the overflow will be brought into memory on the next successful append.

1.4. Memory Buffer Organization

The memory buffer can be considered a sequence of source lines brought in with the A command from a source file. The memory buffer has an associated (imaginary) character pointer CP which moves throughout the memory buffer under command of the operator. The memory buffer appears logically as shown in Figure 3 where the dashes represent characters of the source line of indefinite length, terminated by carriage-return (<cr>) and line-feed (<lf>) characters, and  represents the imaginary character pointer. Note that the CP is always located ahead of the first character of the first line, behind the last character of the last line, or between two characters. The current line CL is the source line which contains the CP.

1.5. Memory Buffer Operation

Upon initiation of ED, the memory buffer is empty (ie, CP is both ahead and behind the first and last character). The operator may either append lines (A command) from the source file, or enter the lines directly from the console with the insert command

I<cr>

ED then accepts any number of input lines, where each line terminates with a <cr> (the <lf> is supplied automatically), until a control-z (denoted by ↑z is typed by the operator. The CP is positioned after the last character entered. The sequence

```
I<cr>
NOW IS THE<cr>
TIME FOR<cr>
ALL GOOD MEN<cr>
↑z
```

leaves the memory buffer as shown below

NOW IS THE<cr><lf>
TIME FOR<cr><lf>
ALL GOOD MEN<cr><lf>



Various commands can then be issued which manipulate the CP or display source text in the vicinity of the CP. The commands shown below with a preceding n indicate that an optional unsigned value can be specified. When preceded by ±, the command can be unsigned, or have an optional preceding plus or minus sign. As before, the pound sign (#) is replaced by 65535. If an integer n is optional, but not supplied, then n=1 is assumed. Finally, if a plus sign is optional, but none is specified, then + is assumed.

- ±B<cr> - move CP to beginning of memory buffer if +, and to bottom if -.
- ±nC<cr> - move CP by ±n characters (toward front of buffer if +), counting the <cr><lf> as two distinct characters
- ±nD<cr> - delete n characters ahead of CP if plus and behind CP if minus.
- ±nK<cr> - kill (ie remove) ±n lines of source text using CP as the current reference. If CP is not at the beginning of the current line when K is issued, then the characters before CP remain if + is specified, while the characters after CP remain if - is given in the command.
- ±nL<cr> - if n=0 then move CP to the beginning of the current line (if it is not already there) if n≠0 then first move the CP to the beginning of the current line, and then move it to the beginning of the line which is n lines down (if +) or up (if -). The CP will stop at the top or bottom of the memory buffer if too large a value of n is specified.

`±nT<cr>` - If `n=0` then type the contents of the current line up to CP. If `n=1` then type the contents of the current line from CP to the end of the line. If `n>1` then type the current line along with `n-1` lines which follow, if `+` is specified. Similarly, if `n>1` and `-` is given, type the previous `n` lines, up to the CP. The break key can be depressed to abort long type-outs.

`±n<cr>` - equivalent to `±nLT`, which moves up or down and types a single line

1.6. Command Strings

Any number of commands can be typed contiguously (up to the capacity of the CP/M console buffer), and are executed only after the `<cr>` is typed. Thus, the operator may use the CP/M console command functions to manipulate the input command:

Rubout	remove the last character
Control-U	delete the entire line
Control-C	re-initialize the CP/M System
Control-E	return carriage for long lines without transmitting buffer (max 128 chars)

Suppose the memory buffer contains the characters shown in the previous section, with the CP following the last character of the buffer. The command strings shown below produce the results shown to the right

<u>Command String</u>	<u>Effect</u>	<u>Resulting Memory Buffer</u>
1. B2T<cr>	move to beginning of buffer and type 2 lines: "NOW IS THE TIME FOR"	NOW IS THE<cr><lf> TIME FOR<cr><lf> ALL GOOD MEN<cr><lf>
2. 5COT<cr>	move CP 5 characters and type the beginning of the line "NOW I"	NOW I  S THE<cr><lf>

- | | | | |
|----|--|---|--|
| 3. | 2L-T<cr> | move two lines down
and type previous
line
"TIME FOR" | NOW IS THE<cr><lf>
TIME FOR<cr><lf>
ALL GOOD MEN<cr><lf> |
| | | | cp |
| 4. | -L#K<cr> | move up one line,
delete 65535 lines
which follow | NOW IS THE<cr><lf> |
| | | | cp |
| 5. | I<cr>
TIME TO<cr>
INSERT<cr>
↑z | insert two lines
of text | NOW IS THE<cr><lf>
TIME TO<cr><lf>
INSERT<cr><lf> |
| | | | cp |
| 6. | -2L#T<cr> | move up two lines,
and type 65535
lines ahead of CP
"NOW IS THE" | NOW IS THE<cr><lf>
TIME TO<cr><lf>
INSERT<cr><lf> |
| | | | cp |
| 7. | <cr> | move down one line
and type one line
"INSERT" | NOW IS THE<cr><lf>
TIME TO<cr><lf>
INSERT<cr><lf> |
| | | | cp |

1.7. Text Search and Alteration

ED also has a command which locates strings within the memory buffer. The command takes the form

$$nF \ c_1 c_2 \dots c_k \ \left\{ \begin{array}{c} <cr> \\ \uparrow z \end{array} \right\}$$

where c_1 through c_k represent the characters to match followed by either a <cr> or control -z*. ED starts at the current position of CP and attempts to match all k characters. The match is attempted n times, and if successful, the CP is moved directly after the character c_k . If the n matches are not successful, the CP is not moved from its initial position. Search strings can include ↑l (control-l), which is replaced by the pair of symbols <cr><lf>.

*The control-z is used if additional commands will be typed following the ↑z.

The following commands illustrate the use of the F command:

<u>Command String</u>	<u>Effect</u>	<u>Resulting Memory Buffer</u>
1. B#T<cr>	move to beginning and type entire buffer	 NOW IS THE<cr><lf> TIME FOR<cr><lf> ALL GOOD MEN<cr><lf>
2. FS T<cr>	find the end of the string "S T"	NOW IS T  HE<cr><lf>
3. FI↑z0TT	find the next "I" and type to the CP then type the remainder of the current line: "TIME FOR"	NOW IS THE<cr><lf> TI  ME FOR<cr><lf> ALL GOOD MEN<cr><lf>

An abbreviated form of the insert command is also allowed, which is often used in conjunction with the F command to make simple textual changes. The form is:

I $c_1 c_2 \dots c_n \uparrow z$ or

I $c_1 c_2 \dots c_n <cr>$

where c_1 through c_n are characters to insert. If the insertion string is terminated by a $\uparrow z$, the characters c_1 through c_n are inserted directly following the CP, and the CP is moved directly after character c_n . The action is the same if the command is followed by a $<cr>$ except that a $<cr><lf>$ is automatically inserted into the text following character c_n . Consider the following command sequences as examples of the F and I commands:

<u>Command String</u>	<u>Effect</u>	<u>Resulting Memory Buffer</u>
BITHIS IS $\uparrow z <cr>$	Insert "THIS IS " at the beginning of the text	THIS IS NOW THE $<cr><lf>$  TIME FOR<cr><lf> ALL GOOD MEN<cr><lf>

FTIME↑z-4DIPLACE↑z<cr>

find "TIME" and delete
it; then insert "PLACE"

THIS IS NOW THE<cr><lf>
PLACE  FOR<cr><lf>
ALL GOOD MEN<cr><lf>

3FO↑z-3D5DICHANGES↑<cr>

find third occurrence
of "O" (ie the second
"O" in GOOD), delete
previous 3 characters;
then insert "CHANGES"

THIS IS NOW THE <cr><lf>
PLACE FOR<cr><lf>
ALL CHANGES  <cr><lf>

-8CISOURCE<cr> move back 8 characters
 and insert the line
 "SOURCE<cr><lf>"

THIS IS NOW THE<cr><lf>
PLACE FOR<cr><lf>
ALL SOURCE<cr><lf>
 CHANGES<cr><lf>

ED also provides a single command which combines the F and I commands to perform simple string substitutions. The command takes the form

$$n \text{ S } c_1 c_2 \dots c_k \uparrow z \ d_1 d_2 \dots d_m \left\{ \begin{array}{c} \text{<cr>} \\ \uparrow z \end{array} \right\}$$

and has exactly the same effect as applying the command string

$$F \ c_1 c_2 \dots c_k \uparrow z - k D I d_1 d_2 \dots d_m \left\{ \begin{array}{c} \text{<cr>} \\ \uparrow z \end{array} \right\}$$

a total of n times. That is, ED searches the memory buffer starting at the current position of CP and successively substitutes the second string for the first string until the end of buffer, or until the substitution has been performed n times.

As a convenience, a command similar to F is provided by ED which automatically appends and writes lines as the search proceeds. The form is

$$n \text{ N } c_1 c_2 \dots c_k \left\{ \begin{array}{c} \text{cr} \\ \uparrow z \end{array} \right\}$$

which searches the entire source file for the nth occurrence of the string $c_1 c_2 \dots c_k$ (recall that F fails if the string cannot be found in the current buffer). The operation of the

N command is precisely the same as F except in the case that the string cannot be found within the current memory buffer. In this case, the entire memory contents is written (ie, an automatic #W is issued). Input lines are then read until the buffer is at least half full, or the entire source file is exhausted. The search continues in this manner until the string has been found n times, or until the source file has been completely transferred to the temporary file.

A final line editing function, called the juxtaposition command takes the form

$$n \ J \ c_1 c_2 \dots c_k \uparrow z \ d_1 d_2 \dots d_m \uparrow z \ e_1 e_2 \dots e_q \left\{ \begin{array}{l} \langle \text{cr} \rangle \\ \uparrow z \end{array} \right\}$$

with the following action applied n times to the memory buffer: search from the current CP for the next occurrence of the string $c_1 c_2 \dots c_k$. If found, insert the string $d_1 d_2 \dots d_m$, and move CP to follow d_m . Then delete all characters following CP up to (but not including) the string $e_1 e_2 \dots e_q$, leaving CP directly after d_m . If $e_1 e_2 \dots e_q$ cannot be found, then no deletion is made. If the current line is

 NOW IS THE TIME<cr><lf>

Then the command

JW $\uparrow z$ WHAT $\uparrow z \uparrow l$ <cr>

Results in

NOW WHAT  <cr><lf>

(Recall that $\uparrow l$ represents the pair <cr><lf> in search and substitute strings).

It should be noted that the number of characters allowed by ED in the F,S,N, and J commands is limited to 100 symbols.

1.8. Source Libraries

ED also allows the inclusion of source libraries during the editing process with the R command. The form of this command is

$R f_1 f_2 \dots f_n \uparrow z$ or

$R f_1 f_2 \dots f_n \langle cr \rangle$

where $f_1 f_2 \dots f_n$ is the name of a source file on the disk with as assumed filetype of 'LIB'. ED reads the specified file, and places the characters into the memory buffer after CP, in a manner similar to the I command. Thus, if the command

RMACRO $\langle cr \rangle$

is issued by the operator, ED reads from the file MACRO.LIB until the end-of-file, and automatically inserts the characters into the memory buffer.

1.9. Repetitive Command Execution

The macro command M allows the ED user to group ED commands together for repeated evaluation. The M command takes the form:

$$n \ M \ c_1 c_2 \dots c_k \left\{ \begin{array}{c} \langle cr \rangle \\ \uparrow z \end{array} \right\}$$

where $c_1 c_2 \dots c_k$ represent a string of ED commands, not including another M command. ED executes the command string n times if $n > 1$. If $n = 0$ or 1, the command string is executed repetitively until an error condition is encountered (e.g., the end of the memory buffer is reached with an F command).

As an example, the following macro changes all occurrences of GAMMA to DELTA within the current buffer, and types each line which is changed:

MFGAMMA $\uparrow z$ -5DIDELTA $\uparrow z$ 0TT $\langle cr \rangle$

or equivalently

MSGAMMA $\uparrow z$ DELTA $\uparrow z$ 0TT $\langle cr \rangle$

2. ED ERROR CONDITIONS

On error conditions, ED prints the last character read before the error, along with an error indicator:

?	unrecognized command
>	memory buffer full (use one of the commands D,K,N,S, or W to remove characters), F,N, or S strings too long.
#	cannot apply command the number of times specified (e.g., in F command)
O	cannot open LIB file in R command

Cyclic redundancy check (CRC) information is written with each output record under CP/M in order to detect errors on subsequent read operations. If a CRC error is detected, CP/M will type

PERM ERR DISK d

where d is the currently selected drive (A,B,...). The operator can choose to ignore the error by typing any character at the console (in this case, the memory buffer data should be examined to see if it was incorrectly read), or the user can reset the system and reclaim the backup file, if it exists. The file can be reclaimed by first typing the contents of the BAK file to ensure that it contains the proper information:

TYPE x.BAK<cr>

where x is the file being edited. Then remove the primary file:

ERA x.y<cr>

and rename the BAK file:

REN x.y=x.BAK<cr>

The file can then be re-edited, starting with the previous version.

3. CONTROL CHARACTERS AND COMMANDS

The following table summarizes the control characters and commands available in ED:

<u>Control Character</u>	<u>Function</u>
↑c	system reboot
↑e	physical <cr><lf> (not actually entered in command)
↑i	logical tab (cols 1,8,15,...)
↑l	logical <cr><lf> in search and substitute strings
↑u	line delete
↑z	string terminator
rubout	character delete
break	discontinue command (e.g., stop typing)

<u>Command</u>	<u>Function</u>
nA	append lines
±B	begin bottom of buffer
±nC	move character positions
±nD	delete characters
E	end edit and close files (normal end)
nF	find string
H	end edit, close and reopen files
I	insert characters
nJ	place strings in juxtaposition
±nK	kill lines
±nL	move down/up lines
nM	macro definition
nN	find next occurrence with autoscan
O	return to original file
±nP	move and print pages
Q	quit with no file changes
R	read library file
nS	substitute strings
±nT	type lines
± U	translate lower to upper case if U, no translation if -U
nW	write lines
nZ	sleep
±n<cr>	move and type (±nLT)
V	VERIFY LINE NUMBERS
ΦV	PRINT MEMORY STATS.

Appendix A: ED 1.4 Enhancements

The ED context editor contains a number of commands which enhance its usefulness in text editing. The improvements are found in the addition of line numbers, free space interrogation, and improved error reporting.

The context editor issued with CP/M 1.4 produces absolute line number prefixes when the "V" (Verify Line Numbers) command is issued. Following the V command, the line number is displayed ahead of each line in the format:

nnnnn:

where nnnnn is an absolute line number in the range 1 to 65535. If the memory buffer is empty, or if the current line is at the end of the memory buffer, then nnnnn appears as 5 blanks.

The user may reference an absolute line number by preceding any command by a number followed by a colon, in the same format as the line number display. In this case, the ED program moves the current line reference to the absolute line number, if the line exists in the current memory buffer. Thus, the command

345:T

is interpreted as "move to absolute line 345, and type the line." Note that absolute line numbers are produced only during the editing process, and are not recorded with the file. In particular, the line numbers will change following a deleted or expanded section of text.

The user may also reference an absolute line number as a backward or forward distance from the current line by preceding the absolute line number by a colon. Thus, the command

:400T

is interpreted as "type from the current line number through the line whose absolute number is 400." Combining the two line reference forms, the command

345::400T

for example, is interpreted as "move to absolute line 345, then type through absolute line 400." Note that absolute line references of this sort can precede any of the standard ED commands.

A special case of the V command, "0V", prints the memory buffer statistics in the form:

free/total

where "free" is the number of free bytes in the memory buffer (in decimal), and "total" is the size of the memory buffer.

ED 1.4 also includes a "block move" facility implemented through the "X" (Xfer) command. The form

nX

transfers the next n lines from the current line to a temporary file called

X\$\$\$\$\$\$\$.LIB

which is active only during the editing process. In general, the user can reposition the current line reference to any portion of the source file and transfer lines to the temporary file. The transferred line accumulate one after another in this file, and can be retrieved by simply typing:

R

which is the trivial case of the library read command. In this case, the entire transferred set of lines is read into the memory buffer. Note that the X command does not remove the transferred lines from the memory buffer, although a K command can be used directly after the X, and the R command does not empty the transferred line file. That is, given that a set of lines has been transferred with the X command, they can be re-read any number of times back into the source file. The command

ØX

is provided, however, to empty the transferred line file.

Note that upon normal completion of the ED program through Q or E, the temporary LIB file is removed. If ED is aborted through cti-C, the LIB file will exist if lines have been transferred, but will generally be empty (a subsequent ED invocation will erase the temporary file).

Due to common typographical errors, ED 1.4 requires several potentially disastrous commands to be typed as single letters, rather than in composite commands. The commands

E (end), H (head), O (original), Q (quit)

must be typed as single letter commands.

ED 1.4 also prints error messages in the form

BREAK "x" AT c

where x is the error character, and c is the command where the error occurred.



Post Office Box 579, Pacific Grove, California 93950, (408) 649-3896

CP/M ASSEMBLER (ASM)
USER'S GUIDE

COPYRIGHT (c) 1976, 1978
DIGITAL RESEARCH

Table of Contents

Section	Page
1. INTRODUCTION	1
2. PROGRAM FORMAT	2
3. FORMING THE OPERAND	4
3.1. Labels	4
3.2. Numeric Constants	4
3.3. Reserved Words	5
3.4. String Constants	6
3.5. Arithmetic and Logical Operators	6
3.6. Precedence of Operators	7
4. ASSEMBLER DIRECTIVES	8
4.1. The ORG Directive	8
4.2. The END Directive	9
4.3. The EQU Directive	9
4.4. The SET Directive	10
4.5. The IF and ENDIF Directives	10
4.6. The DB Directive	11
4.7. The DW Directive	12
5. OPERATION CODES	12
5.1. Jumps, Calls, and Returns	13
5.2. Immediate Operand Instructions	14
5.3. Increment and Decrement Instructions	14
5.4. Data Movement Instructions	14
5.5. Arithmetic Logic Unit Operations	15
5.6. Control Instructions	16
6. ERROR MESSAGES	16
7. A SAMPLE SESSION	17

CP/M Assembler User's Guide

1. INTRODUCTION.

The CP/M assembler reads assembly language source files from the diskette, and produces 8080 machine language in Intel hex format. The CP/M assembler is initiated by typing

```
ASM filename  
or  
ASM filename.parms
```

In both cases, the assembler assumes there is a file on the diskette with the name

```
filename.ASM
```

which contains an 8080 assembly language source file. The first and second forms shown above differ only in that the second form allows parameters to be passed to the assembler to control source file access and hex and print file destinations.

In either case, the CP/M assembler loads, and prints the message

```
CP/M ASSEMBLER VER n.n
```

where n.n is the current version number. In the case of the first command, the assembler reads the source file with assumed file type "ASM" and creates two output files

```
filename.HEX  
and  
filename.PRN
```

the "HEX" file contains the machine code corresponding to the original program in Intel hex format, and the "PRN" file contains an annotated listing showing generated machine code, error flags, and source lines. If errors occur during translation, they will be listed in the PRN file as well as at the console

The second command form can be used to redirect input and output files from their defaults. In this case, the "parms" portion of the command is a three letter group which specifies the origin of the source file, the destination of the hex file, and the destination of the print file. The form is

```
filename.plp2p3
```

where p1, p2, and p3 are single letters

p1: A,B, ..., Y designates the disk name which contains

		the source file
p2:	A,B, ..., Y	designates the disk name which will receive the hex file
	Z	skips the generation of the hex file
p3:	A,B, ..., Y	designates the disk name which will receive the print file
	X	places the listing at the console
	Z	skips generation of the print file

Thus, the command

ASM X.AAA

indicates that the source file (X.ASM) is to be taken from disk A, and that the hex (X.HEX) and print (X.PRN) files are to be created also on disk A. This form of the command is implied if the assembler is run from disk A. That is, given that the operator is currently addressing disk A, the above command is equivalent to

ASM X

The command

ASM X.ABX

indicates that the source file is to be taken from disk A, the hex file is placed on disk B, and the listing file is to be sent to the console. The command

ASM X.BZZ

takes the source file from disk B, and skips the generation of the hex and print files (this command is useful for fast execution of the assembler to check program syntax).

The source program format is compatible with both the Intel 8080 assembler (macros are not currently implemented in the CP/M assembler, however), as well as the Processor Technology Software Package #1 assembler. That is, the CP/M assembler accepts source programs written in either format. There are certain extensions in the CP/M assembler which make it somewhat easier to use. These extensions are described below.

2. PROGRAM FORMAT.

An assembly language program acceptable as input to the assembler consists of a sequence of statements of the form

line# label operation operand ;comment

where any or all of the fields may be present in a particular instance. Each

assembly language statement is terminated with a carriage return and line feed (the line feed is inserted automatically by the ED program), or with the character "!" which is treated as an end-of-line by the assembler (thus, multiple assembly language statements can be written on the same physical line if separated by exclaim symbols).

The line# is an optional decimal integer value representing the source program line number, which is allowed on any source line to maintain compatibility with the Processor Technology format. In general, these line numbers will be inserted if a line-oriented editor is used to construct the original program, and thus ASM ignores this field if present.

The label field takes the form

 identifier
or
 identifier:

and is optional, except where noted in particular statement types. The identifier is a sequence of alphanumeric characters (alphabetic and numbers), where the first character is alphabetic. Identifiers can be freely used by the programmer to label elements such as program steps and assembler directives, but cannot exceed 16 characters in length. All characters are significant in an identifier, except for the embedded dollar symbol (\$) which can be used to improve readability of the name. Further, all lower case alphabetic characters are treated as if they were upper case. Note that the ":" following the identifier in a label is optional (to maintain compatibility between Intel and Processor Technology). Thus, the following are all valid instances of labels

x	xy	long\$name
x:	yx1:	longer\$name'data:
xly2	xlx2	x234\$5678\$9012\$3456:

The operation field contains either an assembler directive, or pseudo operation, or an 8080 machine operation code. The pseudo operations and machine operation codes are described below.

The operand field of the statement, in general, contains an expression formed out of constants and labels, along with arithmetic and logical operations on these elements. Again, the complete details of properly formed expressions are given below.

The comment field contains arbitrary characters following the ";" symbol until the next real or logical end-of-line. These characters are read, listed, and otherwise ignored by the assembler. In order to maintain compatibility with the Processor Technology assembler, the CP/M assembler also treat statements which begin with a "*" in column one as comment statements, which are listed and ignored in the assembly process. Note that the Processor

Technology assembler has the side effect in its operation of ignoring the characters after the operand field has been scanned. This causes an ambiguous situation when attempting to be compatible with Intel's language, since arbitrary expressions are allowed in this case. Hence, programs which use this side effect to introduce comments, must be edited to place a ";" before these fields in order to assemble correctly.

The assembly language program is formulated as a sequence of statements of the above form, terminated optionally by an END statement. All statements following the END are ignored by the assembler.

3. FORMING THE OPERAND.

In order to completely describe the operation codes and pseudo operations, it is necessary to first present the form of the operand field, since it is used in nearly all statements. Expressions in the operand field consist of simple operands (labels, constants, and reserved words), combined in properly formed subexpressions by arithmetic and logical operators. The expression computation is carried out by the assembler as the assembly proceeds. Each expression must produce a 16-bit value during the assembly. Further, the number of significant digits in the result must not exceed the intended use. That is, if an expression is to be used in a byte move immediate instruction, then the most significant 8 bits of the expression must be zero. The restrictions on the expression significance is given with the individual instructions.

3.1. Labels.

As discussed above, a label is an identifier which occurs on a particular statement. In general, the label is given a value determined by the type of statement which it precedes. If the label occurs on a statement which generates machine code or reserves memory space (e.g, a MOV instruction, or a DS pseudo operation), then the label is given the value of the program address which it labels. If the label precedes an EQU or SET, then the label is given the value which results from evaluating the operand field. Except for the SET statement, an identifier can label only one statement.

When a label appears in the operand field, its value is substituted by the assembler. This value can then be combined with other operands and operators to form the operand field for a particular instruction.

3.2. Numeric Constants.

A numeric constant is a 16-bit value in one of several bases. The base, called the radix of the constant, is denoted by a trailing radix indicator. The radix indicators are

B	binary constant (base 2)
O	octal constant (base 8)

Q	octal constant (base 8)
D	decimal constant (base 10)
H	hexadecimal constant (base 16)

Q is an alternate radix indicator for octal numbers since the letter O is easily confused with the digit 0. Any numeric constant which does not terminate with a radix indicator is assumed to be a decimal constant.

A constant is thus composed as a sequence of digits, followed by an optional radix indicator, where the digits are in the appropriate range for the radix. That is binary constants must be composed of 0 and 1 digits, octal constants can contain digits in the range 0 - 7, while decimal constants contain decimal digits. Hexadecimal constants contain decimal digits as well as hexadecimal digits A (10D), B (11D), C (12D), D (13D), E (14D), and F (15D). Note that the leading digit of a hexadecimal constant must be a decimal digit in order to avoid confusing a hexadecimal constant with an identifier (a leading 0 will always suffice). A constant composed in this manner must evaluate to a binary number which can be contained within a 16-bit counter, otherwise it is truncated on the right by the assembler. Similar to identifiers, imbedded "\$" are allowed within constants to improve their readability. Finally, the radix indicator is translated to upper case if a lower case letter is encountered. The following are all valid instances of numeric constants

1234	1234D	1100B	1111\$0000\$1111\$0000B
1234H	0FFEh	3377O	33\$77\$22Q
3377o	0fe3h	1234d	0ffffh

3.3. Reserved Words.

There are several reserved character sequences which have predefined meanings in the operand field of a statement. The names of 8080 registers are given below, which, when encountered, produce the value shown to the right

A	7
B	0
C	1
D	2
E	3
H	4
L	5
M	6
SP	6
PSW	6

(again, lower case names have the same values as their upper case equivalents). Machine instructions can also be used in the operand field, and evaluate to their internal codes. In the case of instructions which require operands, where the specific operand becomes a part of the binary bit pattern

of the instruction (e.g, MOV A,B), the value of the instruction (in this case MOV) is the bit pattern of the instruction with zeroes in the optional fields (e.g, MOV produces 40H).

When the symbol "\$" occurs in the operand field (not imbedded within identifiers and numeric constants) its value becomes the address of the next instruction to generate, not including the instruction contained within the current logical line.

3.4. String Constants.

String constants represent sequences of ASCII characters, and are represented by enclosing the characters within apostrophe symbols ('). All strings must be fully contained within the current physical line (thus allowing "!" symbols within strings), and must not exceed 64 characters in length. The apostrophe character itself can be included within a string by representing it as a double apostrophe (the two keystrokes ``'), which becomes a single apostrophe when read by the assembler. In most cases, the string length is restricted to either one or two characters (the DB pseudo operation is an exception), in which case the string becomes an 8 or 16 bit value, respectively. Two character strings become a 16-bit constant, with the second character as the low order byte, and the first character as the high order byte.

The value of a character is its corresponding ASCII code. There is no case translation within strings, and thus both upper and lower case characters can be represented. Note however, that only graphic (printing) ASCII characters are allowed within strings. Valid strings are

```
'A'      'AB'      'ab'      'c'
.....  'a
'Walla Walla Wash.'
'She said "Hello" to me.'
'I said "Hello" to her.'
```

3.5. Arithmetic and Logical Operators.

The operands described above can be combined in normal algebraic notation using any combination of properly formed operands, operators, and parenthesized expressions. The operators recognized in the operand field are

a + b	unsigned arithmetic sum of a and b
a - b	unsigned arithmetic difference between a and b
+ b	unary plus (produces b)
- b	unary minus (identical to 0 - b)
a * b	unsigned magnitude multiplication of a and b
a / b	unsigned magnitude division of a by b
a MOD b	remainder after a / b
NOT b	logical inverse of b (all 0's become 1's, 1's become 0's), where b is considered a 16-bit value

a AND b	bit-by-bit logical and of a and b
a OR b	bit-by-bit logical or of a and b
a XOR b	bit-by-bit logical exclusive or of a and b
a SHL b	the value which results from shifting a to the left by an amount b, with zero fill
a SHR b	the value which results from shifting a to the right by an amount b, with zero fill

In each case, a and b represent simple operands (labels, numeric constants, reserved words, and one or two character strings), or fully enclosed parenthesized subexpressions such as

```

10+20      10h+37Q      L1 /3      (L2+4) SHR 3
('a' and 5fh) + '0'      ('B'+B) OR (PSW+M)
(1+(2+c)) shr (A-(B+1))

```

Note that all computations are performed at assembly time as 16-bit unsigned operations. Thus, -1 is computed as 0-1 which results in the value 0ffffh (i.e., all 1's). The resulting expression must fit the operation code in which it is used. If, for example, the expression is used in a ADI (add immediate) instruction, then the high order eight bits of the expression must be zero. As a result, the operation "ADI -1" produces an error message (-1 becomes 0ffffh which cannot be represented as an 8 bit value), while "ADI (-1) AND 0FFH" is accepted by the assembler since the "AND" operation zeroes the high order bits of the expression.

3.6. Precedence of Operators.

As a convenience to the programmer, ASM assumes that operators have a relative precedence of application which allows the programmer to write expressions without nested levels of parentheses. The resulting expression has assumed parentheses which are defined by the relative precedence. The order of application of operators in unparenthesized expressions is listed below. Operators listed first have highest precedence (they are applied first in an unparenthesized expression), while operators listed last have lowest precedence. Operators listed on the same line have equal precedence, and are applied from left to right as they are encountered in an expression

```

* / MOD SHL SHR
- +
NOT
AND
OR XOR

```

Thus, the expressions shown to the left below are interpreted by the assembler as the fully parenthesized expressions shown to the right below

a * b + c	(a * b) + c
a + b * c	a + (b * c)
a MOD b * c SHL d	((a MOD b) * c) SHL d

a OR b AND NOT c + d SHL e a OR (b AND (NOT (c + (d SHL e))))

Balanced parenthesized subexpressions can always be used to override the assumed parentheses, and thus the last expression above could be rewritten to force application of operators in a different order as

(a OR b) AND (NOT c) + d SHL e

resulting in the assumed parentheses

(a OR b) AND ((NOT c) + (d SHL e))

Note that an unparenthesized expression is well-formed only if the expression which results from inserting the assumed parentheses is well-formed.

4. ASSEMBLER DIRECTIVES.

Assembler directives are used to set labels to specific values during the assembly, perform conditional assembly, define storage areas, and specify starting addresses in the program. Each assembler directive is denoted by a "pseudo operation" which appears in the operation field of the line. The acceptable pseudo operations are

ORG	set the program or data origin
END	end program, optional start address
EQU	numeric "equate"
SET	numeric "set"
IF	begin conditional assembly
ENDIF	end of conditional assembly
DB	define data bytes
DW	define data words
DS	define data storage area

The individual pseudo operations are detailed below

4.1. The ORG directive.

The ORG statement takes the form

label ORG expression

where "label" is an optional program label, and expression is a 16-bit expression, consisting of operands which are defined previous to the ORG statement. The assembler begins machine code generation at the location specified in the expression. There can be any number of ORG statements within a particular program, and there are no checks to ensure that the programmer is not defining overlapping memory areas. Note that most programs written for the CP/M system begin with an ORG statement of the form

ORG 100H

which causes machine code generation to begin at the base of the CP/M transient program area. If a label is specified in the ORG statement, then the label is given the value of the expression (this label can then be used in the operand field of other statements to represent this expression).

4.2. The END directive.

The END statement is optional in an assembly language program, but if it is present it must be the last statement (all subsequent statements are ignored in the assembly). The two forms of the END directive are

```
label    END
label    END    expression
```

where the label is again optional. If the first form is used, the assembly process stops, and the default starting address of the program is taken as 0000. Otherwise, the expression is evaluated, and becomes the program starting address (this starting address is included in the last record of the Intel formatted machine code "hex" file which results from the assembly). Thus, most CP/M assembly language programs end with the statement

```
END 100H
```

resulting in the default starting address of 100H (beginning of the transient program area).

4.3. The EQU directive.

The EQU (equate) statement is used to set up synonyms for particular numeric values. the form is

```
label    EQU    expression
```

where the label must be present, and must not label any other statement. The assembler evaluates the expression, and assigns this value to the identifier given in the label field. The identifier is usually a name which describes the value in a more human-oriented manner. Further, this name is used throughout the program to "parameterize" certain functions. Suppose for example, that data received from a Teletype appears on a particular input port, and data is sent to the Teletype through the next output port in sequence. The series of equate statements could be used to define these ports for a particular hardware environment

```
TTYBASE    EQU    10H        ;BASE PORT NUMBER FOR TTY
TTYIN      EQU    TTYBASE    ;TTY DATA IN
TTYOUT     EQU    TTYBASE+1  ;TTY DATA OUT
```

At a later point in the program, the statements which access the Teletype could appear as

```

IN    TTYIN    ;READ TTY DATA TO REG-A
...
OUT   TTYOUT   ;WRITE DATA TO TTY FROM REG-A

```

making the program more readable than if the absolute i/o ports had been used. Further, if the hardware environment is redefined to start the Teletype communications ports at 7FH instead of 10H, the first statement need only be changed to

```

TTYBASE EQU 7FH ;BASE PORT NUMBER FOR TTY

```

and the program can be reassembled without changing any other statements.

4.4. The SET Directive.

The SET statement is similar to the EQU, taking the form

```

label SET expression

```

except that the label can occur on other SET statements within the program. The expression is evaluated and becomes the current value associated with the label. Thus, the EQU statement defines a label with a single value, while the SET statement defines a value which is valid from the current SET statement to the point where the label occurs on the next SET statement. The use of the SET is similar to the EQU statement, but is used most often in controlling conditional assembly.

4.5. The IF and ENDIF directives.

The IF and ENDIF statements define a range of assembly language statements which are to be included or excluded during the assembly process. The form is

```

IF expression
statement#1
statement#2
...
statement#n
ENDIF

```

Upon encountering the IF statement, the assembler evaluates the expression following the IF (all operands in the expression must be defined ahead of the IF statement). If the expression evaluates to a non-zero value, then statement#1 through statement#n are assembled; if the expression evaluates to zero, then the statements are listed but not assembled. Conditional assembly is often used to write a single "generic" program which includes a number of possible run-time environments, with only a few specific portions of the program selected for any particular assembly. The following program segments for example, might be part of a program which communicates with either a Teletype or a CRT console (but not both) by selecting a particular value for TTY before the assembly begins

```

TRUE    EQU    0FFFFH    ;DEFINE VALUE OF TRUE
FALSE   EQU    NOT TRUE  ;DEFINE VALUE OF FALSE
;
TTY      EQU    TRUE      ;TRUE IF TTY, FALSE IF CRT
;
TTYBASE  EQU    10H       ;BASE OF TTY I/O PORTS
CRIBASE  EQU    20H       ;BASE OF CRT I/O PORTS
        IF      TTY       ;ASSEMBLE RELATIVE TO TTYBASE
CONIN     EQU    TTYBASE   ;CONSOLE INPUT
CONOUT    EQU    TTYBASE+1 ;CONSOLE OUTPUT
        ENDIF
;
        IF      NOT TTY    ;ASSEMBLE RELATIVE TO CRIBASE
CONIN     EQU    CRIBASE   ;CONSOLE INPUT
CONOUT    EQU    CRIBASE+1 ;CONSOLE OUTPUT
        ENDIF
        ...
        IN      CONIN      ;READ CONSOLE DATA
        ...
        OUT     CONOUT     ;WRITE CONSOLE DATA

```

In this case, the program would assemble for an environment where a Teletype is connected, based at port 10H. The statement defining TTY could be changed to

```
TTY      EQU    FALSE
```

and, in this case, the program would assemble for a CRT based at port 20H.

4.6. The DB Directive.

The DB directive allows the programmer to define initialize storage areas in single precision (byte) format. The statement form is

```
label    DB    e#1, e#2, ..., e#n
```

where e#1 through e#n are either expressions which evaluate to 8-bit values (the high order eight bits must be zero), or are ASCII strings of length no greater than 64 characters. There is no practical restriction on the number of expressions included on a single source line. The expressions are evaluated and placed sequentially into the machine code file following the last program address generated by the assembler. String characters are similarly placed into memory starting with the first character and ending with the last character. Strings of length greater than two characters cannot be used as operands in more complicated expressions (i.e., they must stand alone between the commas). Note that ASCII characters are always placed in memory with the parity bit reset (0). Further, recall that there is no translation from lower to upper case within strings. The optional label can be used to reference the data area throughout the remainder of the program. Examples of

valid DB statements are

```
data:  DB  0,1,2,3,4,5
        DB  data and 0ffh,5,3770,1+2+3+4
signon: DB  'please type your name',cr,lf,0
        DB  'AB' SHR 8, 'C', 'DE' AND 7FH
```

4.7. The DW Directive.

The DW statement is similar to the DB statement except double precision (two byte) words of storage are initialized. The form is

```
label  DW  e#1, e#2, ..., e#n
```

where e#1 through e#n are expressions which evaluate to 16-bit results. Note that ASCII strings of length one or two characters are allowed, but strings longer than two characters disallowed. In all cases, the data storage is consistent with the 8080 processor: the least significant byte of the expression is stored first in memory, followed by the most significant byte. Examples are

```
doub:  DW  0ffefh,doub+4,signon-$,255+255
        DW  'a', 5, 'ab', 'CD', 6 shl 8 or 11b
```

4.8. The DS Directive.

The DS statement is used to reserve an area of uninitialized memory, and takes the form

```
label  DS  expression
```

where the label is optional. The assembler begins subsequent code generation after the area reserved by the DS. Thus, the DS statement given above has exactly the same effect as the statement

```
label:  EQU  $  ;LABEL VALUE IS CURRENT CODE LOCATION
        ORG  $+expression ;MOVE PAST RESERVED AREA
```

5. OPERATION CODES.

Assembly language operation codes form the principal part of assembly language programs, and form the operation field of the instruction. In general, ASM accepts all the standard mnemonics for the Intel 8080 microcomputer, which are given in detail in the Intel manual "8080 Assembly Language Programming Manual." Labels are optional on each input line and, if included, take the value of the instruction address immediately before the instruction is issued. The individual operators are listed briefly in the

following sections for completeness, although it is understood that the Intel manuals should be referenced for exact operator details. In each case,

e3 represents a 3-bit value in the range 0-7
 which can be one of the predefined registers
 A, B, C, D, E, H, L, M, SP, or PSW.

e8 represents an 8-bit value in the range 0-255

e16 represents a 16-bit value in the range 0-65535

which can themselves be formed from an arbitrary combination of operands and operators. In some cases, the operands are restricted to particular values within the allowable range, such as the PUSH instruction. These cases will be noted as they are encountered.

In the sections which follow, each operation codes is listed in its most general form, along with a specific example, with a short explanation and special restrictions.

5.1. Jumps, Calls, and Returns.

The Jump, Call, and Return instructions allow several different forms which test the condition flags set in the 8080 microcomputer CPU. The forms are

JMP e16	JMP L1	Jump unconditionally to label
JNZ e16	JMP L2	Jump on non zero condition to label
JZ e16	JMP 100H	Jump on zero condition to label
JNC e16	JNC L1+4	Jump no carry to label
JC e16	JC L3	Jump on carry to label
JPO e16	JPO \$+8	Jump on parity odd to label
JPE e16	JPE L4	Jump on even parity to label
JP e16	JP GAMMA	Jump on positive result to label
JM e16	JM a1	Jump on minus to label
CALL e16	CALL S1	Call subroutine unconditionally
CNZ e16	CNZ S2	Call subroutine if non zero flag
CZ e16	CZ 100H	Call subroutine on zero flag
CNC e16	CNC S1+4	Call subroutine if no carry set
CC e16	CC S3	Call subroutine if carry set
CPO e16	CPO \$+8	Call subroutine if parity odd
CPE e16	CPE S4	Call subroutine if parity even
CP e16	CP GAMMA	Call subroutine if positive result
CM e16	CM b1\$c2	Call subroutine if minus flag
RST e3	RST 0	Programmed "restart", equivalent to CALL 8*e3, except one byte call

RET	Return from subroutine
RNZ	Return if non zero flag set
RZ	Return if zero flag set
RNC	Return if no carry
RC	Return if carry flag set
RPO	Return if parity is odd
RPE	Return if parity is even
RP	Return if positive result
RM	Return if minus flag is set

5.2. Immediate Operand Instructions.

Several instructions are available which load single or double precision registers, or single precision memory cells, with constant values, along with instructions which perform immediate arithmetic or logical operations on the accumulator (register A).

MVI e3,e8	MVI B,255	Move immediate data to register A, B, C, D, E, H, L, or M (memory)
ADI e8	ADI 1	Add immediate operand to A without carry
ACI e8	ACI 0FFH	Add immediate operand to A with carry
SUI e8	SUI L + 3	Subtract from A without borrow (carry)
SBI e8	SBI L AND 11B	Subtract from A with borrow (carry)
ANI e8	ANI \$ AND 7FH	Logical "and" A with immediate data
XRI e8	XRI 1111\$0000B	"Exclusive or" A with immediate data
ORI e8	ORI L AND 1+1	Logical "or" A with immediate data
CPI e8	CPI 'a'	Compare A with immediate data (same as SUI except register A not changed)
LXI e3,e16	LXI B,100H	Load extended immediate to register pair (e3 must be equivalent to B,D,H, or SP)

5.3. Increment and Decrement Instructions.

Instructions are provided in the 8080 repertoire for incrementing or decrementing single and double precision registers. The instructions are

INR e3	INR E	Single precision increment register (e3 produces one of A, B, C, D, E, H, L, M)
DCR e3	DCR A	Single precision decrement register (e3 produces one of A, B, C, D, E, H, L, M)
INX e3	INX SP	Double precision increment register pair (e3 must be equivalent to B,D,H, or SP)
DCX e3	DCX B	Double precision decrement register pair (e3 must be equivalent to B,D,H, or SP)

5.4. Data Movement Instructions.

Instructions which move data from memory to the CPU and from CPU to memory are given below

MOV e3,e3	MOV A,B	Move data to leftmost element from rightmost element (e3 produces one of A,B,C,D,E,H,L, or M). MOV M,M is disallowed
LDAX e3	LDAX B	Load register A from computed address (e3 must produce either B or D)
STAX e3	STAX D	Store register A to computed address (e3 must produce either B or D)
LHLD e16	LHLD L1	Load HL direct from location e16 (double precision load to H and L)
SHLD e16	SHLD L5+x	Store HL direct to location e16 (double precision store from H and L to memory)
LDA e16	LDA Gamma	Load register A from address e16
STA e16	STA X3-5	Store register A into memory at e16
POP e3	POP PSW	Load register pair from stack, set SP (e3 must produce one of B, D, H, or PSW)
PUSH e3	PUSH B	Store register pair into stack, set SP (e3 must produce one of B, D, H, or PSW)
IN e8	IN 0	Load register A with data from port e8
OUT e8	OUT 255	Send data from register A to port e8
XTHL		Exchange data from top of stack with HL
PCHL		Fill program counter with data from HL
SPHL		Fill stack pointer with data from HL
XCHG		Exchange DE pair with HL pair

5.5. Arithmetic Logic Unit Operations.

Instructions which act upon the single precision accumulator to perform arithmetic and logic operations are

ADD e3	ADD B	Add register given by e3 to accumulator without carry (e3 must produce one of A, B, C, D, E, H, or L)
ADC e3	ADC L	Add register to A with carry, e3 as above
SUB e3	SUB H	Subtract reg e3 from A without carry, e3 is defined as above
SBB e3	SBB 2	Subtract register e3 from A with carry, e3 defined as above
ANA e3	ANA 1+1	Logical "and" reg with A, e3 as above
XRA e3	XRA A	"Exclusive or" with A, e3 as above
ORA e3	ORA B	Logical "or" with A, e3 defined as above
CMP e3	CMP H	Compare register with A, e3 as above
DAA		Decimal adjust register A based upon last arithmetic logic unit operation
CMA		Complement the bits in register A
STC		Set the carry flag to 1

CMC	Complement the carry flag
RLC	Rotate bits left, (re)set carry as a side effect (high order A bit becomes carry)
RRC	Rotate bits right, (re)set carry as side effect (low order A bit becomes carry)
RAL	Rotate carry/A register to left (carry is involved in the rotate)
RAR	Rotate carry/A register to right (carry is involved in the rotate)

DAD e3	DAD B	Double precision add register pair e3 to HL (e3 must produce B, D, H, or SP)
--------	-------	--

5.6. Control Instructions.

The four remaining instructions are categorized as control instructions, and are listed below

HLT	Halt the 8080 processor
DI	Disable the interrupt system
EI	Enable the interrupt system
NOP	No operation

6. ERROR MESSAGES.

When errors occur within the assembly language program, they are listed as single character flags in the leftmost position of the source listing. The line in error is also echoed at the console so that the source listing need not be examined to determine if errors are present. The error codes are

D	Data error: element in data statement cannot be placed in the specified data area
E	Expression error: expression is ill-formed and cannot be computed at assembly time
L	Label error: label cannot appear in this context (may be duplicate label)
N	Not implemented: features which will appear in future ASM versions (e.g., macros) are recognized, but flagged in this version)
O	Overflow: expression is too complicated (i.e., too many pending operators) to computed, simplify it
P	Phase error: label does not have the same value on two subsequent passes through the program

- R Register error: the value specified as a register
 is not compatible with the operation code
- V Value error: operand encountered in expression is
 improperly formed

Several error message are printed which are due to terminal error conditions

- | | |
|-------------------------|--|
| NO SOURCE FILE PRESENT | The file specified in the ASM command does not exist on disk |
| NO DIRECTORY SPACE | The disk directory is full, erase files which are not needed, and retry |
| SOURCE FILE NAME ERROR | Improperly formed ASM file name (e.g., it is specified with "?" fields) |
| SOURCE FILE READ ERROR | Source file cannot be read properly by the assembler, execute a TYPE to determine the point of error |
| OUTPUT FILE WRITE ERROR | Output files cannot be written properly, most likely cause is a full disk, erase and retry |
| CANNOT CLOSE FILE | Output file cannot be closed, check to see if disk is write protected |

7. A SAMPLE SESSION.

The following session shows interaction with the assembler and debugger in the development of a simple assembly language program.

ASM SORT, assemble SORT.ASM

CP/M ASSEMBLER - VER 1.0

015C next free address
003H USE FACTOR % of table used 00 To FF (hexadecimal)
END OF ASSEMBLY

DIR SORT.*

SORT ASM source file
SORT BAK backup from last edit
SORT PRN print file (contains tab characters)
SORT HEX machine code file
A>TYPE SORT.PRN

Source line

machine code location:

0100	← generated machine code	SORT PROGRAM IN CP/M ASSEMBLY LANGUAGE START AT THE BEGINNING OF THE TRANSIENT PROGRAM AREA ORG 100H
0100 214601	SORT:	LXI H, SW ; ADDRESS SWITCH TOGGLE
0103 3601		MVI M, 1 ; SET TO 1 FOR FIRST ITERATION
0105 214701		LXI H, I ; ADDRESS INDEX
0108 3600		MVI M, 0 ; I = 0
;		
010A 7E	COMP:	COMPARE I WITH ARRAY SIZE
010B FE09		MOV A, M ; A REGISTER = I
010D D21901		CPI N-1 ; CY SET IF I < (N-1)
		JNC CONT ; CONTINUE IF I <= (N-2)
;		
0110 214601		END OF ONE PASS THROUGH DATA
0113 7EB7C20001		LXI H, SW ; CHECK FOR ZERO SWITCHES
		MOV A, M! ORA A! JNZ SORT ; END OF SORT IF SW=0
0118 FF		RST 7 ; GO TO THE DEBUGGER INSTEAD OF REP
;		
0119 5F16002148	CONT:	CONTINUE THIS PASS
0121 4E792346		ADDRESSING I, SO LOAD AV(I) INTO REGISTERS
		MOV E, A! MVI D, 0! LXI H, AV! DAD D! DAD D
		MOV C, M! MOV A, C! INX H! MOV B, M
		LOW ORDER BYTE IN A AND C, HIGH ORDER BYTE IN B
;		
0125 23		MOV H AND L TO ADDRESS AV(I+1)
		INX H
;		
0126 965778239E		COMPARE VALUE WITH REGS CONTAINING AV(I)
		SUB M! MOV D, A! MOV A, B! INX H! SBB M ; SUBTRACT
;		
012B DA3F01		BORROW SET IF AV(I+1) > AV(I)
		JC INCI ; SKIP IF IN PROPER ORDER
;		
012E B2CA3F01		CHECK FOR EQUAL VALUES
		ORA D! JZ INCI ; SKIP IF AV(I) = AV(I+1)

```
0132 56702B5E      MOV D,M! MOV M,B! DCX H! MOV E,M
0136 712B722B73    MOV M,C! DCX H! MOV M,D! DCX H! MOV M,E
```

```
013B 21460134      INCREMENT SWITCH COUNT
LXI H,SW! INR M
```

```
013F 21470134C3INC1: INCREMENT I
LXI H,I! INR M! JMP COMP
```

```
0146 00      SW: DB 0 ; RESERVE SPACE FOR SWITCH COUNT
0147      I: DS 1 ; SPACE FOR INDEX
0148 050064001EAV: DW 5,100,30,50,20,7,1000,300,100,-32767
000A = N EQU ($-AV)/2 ; COMPUTE N INSTEAD OF PRE
015C ← equate value END
```

A>TYPE SORT.HEX

```
:10010000214601360121470136007EFE09D2190140
:100110002146017EB7C20001FF5F16002148011900
:10012000194E79234623965778239EDA3F01B2CAA7
:100130003F0156702B5E712B722B732146013421C7
:07014000470134C30A01006E
:10014800050064001E00320014000700E8032C01BB
:0401580064000180BE
:0000000000
```

machine code in
Hex format

A>DDT SORT.HEX, start debug run

16K DDT VER 1.0

NEXT PC

015C 0000 default address (no address on END statement)

-XP

P=0000 100, Change PC to 100

-UFFFF, untrace for 65535 steps

C0Z0M0E010 A=00 B=0000 D=0000 H=0000 S=0100 P=0100 LXI H,0146+0100

-T10, trace 10 steps

```
C0Z0M0E010 A=01 B=0000 D=0000 H=0146 S=0100 P=0100 LXI H,0146
C0Z0M0E010 A=01 B=0000 D=0000 H=0146 S=0100 P=0103 MVI M,01
C0Z0M0E010 A=01 B=0000 D=0000 H=0146 S=0100 P=0105 LXI H,0147
C0Z0M0E010 A=01 B=0000 D=0000 H=0147 S=0100 P=0108 MVI M,00
C0Z0M0E010 A=01 B=0000 D=0000 H=0147 S=0100 P=010A MOV A,M
C0Z0M0E010 A=00 B=0000 D=0000 H=0147 S=0100 P=010B CPI 09
C1Z0M1E010 A=00 B=0000 D=0000 H=0147 S=0100 P=010D JNC 0119
C1Z0M1E010 A=00 B=0000 D=0000 H=0147 S=0100 P=0110 LXI H,0146
C1Z0M1E010 A=00 B=0000 D=0000 H=0146 S=0100 P=0113 MOV A,M
C1Z0M1E010 A=01 B=0000 D=0000 H=0146 S=0100 P=0114 ORA A
C0Z0M0E010 A=01 B=0000 D=0000 H=0146 S=0100 P=0115 JNZ 0100
C0Z0M0E010 A=01 B=0000 D=0000 H=0146 S=0100 P=0100 LXI H,0146
C0Z0M0E010 A=01 B=0000 D=0000 H=0146 S=0100 P=0103 MVI M,01
C0Z0M0E010 A=01 B=0000 D=0000 H=0146 S=0100 P=0105 LXI H,0147
C0Z0M0E010 A=01 B=0000 D=0000 H=0147 S=0100 P=0108 MVI M,00
C0Z0M0E010 A=01 B=0000 D=0000 H=0147 S=0100 P=010A MOV A,M*010B
-A10D
```

abort with
rubout

010D JC 119, change to a jump on carry

stopped at
10BH

-XP₂

P=0108 100₂ reset program counter back to beginning of program

-T10₂ trace execution for 10H steps

```
C0Z0M0E010 A=00 B=0000 D=0000 H=0147 S=0100 P=0100 LXI H,0146
C0Z0M0E010 A=00 B=0000 D=0000 H=0146 S=0100 P=0103 MVI M,01
C0Z0M0E010 A=00 B=0000 D=0000 H=0146 S=0100 P=0105 LXI H,0147
C0Z0M0E010 A=00 B=0000 D=0000 H=0147 S=0100 P=0108 MVI M,00
C0Z0M0E010 A=00 B=0000 D=0000 H=0147 S=0100 P=010A MOV A,M
C0Z0M0E010 A=00 B=0000 D=0000 H=0147 S=0100 P=010B CPI 09
C1Z0M1E010 A=00 B=0000 D=0000 H=0147 S=0100 P=010D JC 0119
C1Z0M1E010 A=00 B=0000 D=0000 H=0147 S=0100 P=0119 MOV E,A
C1Z0M1E010 A=00 B=0000 D=0000 H=0147 S=0100 P=011A MVI D,00
C1Z0M1E010 A=00 B=0000 D=0000 H=0147 S=0100 P=011C LXI H,0148
C1Z0M1E010 A=00 B=0000 D=0000 H=0148 S=0100 P=011F DAD D
C0Z0M1E010 A=00 B=0000 D=0000 H=0148 S=0100 P=0120 DAD D
C0Z0M1E010 A=00 B=0000 D=0000 H=0148 S=0100 P=0121 MOV C,M
C0Z0M1E010 A=00 B=0005 D=0000 H=0148 S=0100 P=0122 MOV A,C
C0Z0M1E010 A=05 B=0005 D=0000 H=0148 S=0100 P=0123 INX H
C0Z0M1E010 A=05 B=0005 D=0000 H=0149 S=0100 P=0124 MOV B,M*0125
```

-L100₂

```
0100 LXI H,0146
0103 MVI M,01
0105 LXI H,0147
0108 MVI M,00
010A MOV A,M
010B CPI 09
010D JC 0119
0110 LXI H,0146
0113 MOV A,M
0114 ORA A
0115 JNZ 0100
```

} list some code
from 100H

Automatic
breakpoint

-L₂

```
0118 RST 07
0119 MOV E,A
011A MVI D,00
011C LXI H,0148
```

} list more

- abort list with rubart

-G,118₂ start program from current PC (0125H) and run in real time to 11BH

*0127 stopped with an external interrupt 7 from front panel (program was looping indefinitely)

-T⁴₂ look at looping program in trace mode

```
C0Z0M0E010 A=38 B=0064 D=0006 H=0156 S=0100 P=0127 MOV D,A
C0Z0M0E010 A=38 B=0064 D=3806 H=0156 S=0100 P=0128 MOV A,B
C0Z0M0E010 A=00 B=0064 D=3806 H=0156 S=0100 P=0129 INX H
C0Z0M0E010 A=00 B=0064 D=3806 H=0157 S=0100 P=012A SBB M*012B
-D148
```

data is sorted, but program doesn't stop.

```
0148 05 00 07 00 14 00 1E 00 .....
0150 32 00 64 00 64 00 2C 01 E8 03 01 80 00 00 00 00 2.D.D.....
0160 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
```

-G, return to CP/M

DDT SORT.HEX, reload the memory image

16K DDT VER 1.0

NEXT PC

015C 0000

-XP

P=0000 100, Set PC to beginning of program

-L10D, list bad opcode

010D JNC 0119 ✓

0110 LXI H, 0146

- abort list with rubout

-A10D, assemble new opcode

010D JC 119,

0110,

-L100, list starting section of program

0100 LXI H, 0146

0103 MVI M, 01

0105 LXI H, 0147

0108 MVI M, 00

- abort list with rubout

-A103, change "switch" initialization to 00

0103 MVI M, 0,

0105,

-c return to CP/M with ctrl-C (G works as well)

SAVE 1 SORT.COM, save 1 page (256 bytes, from 100H to 1FFH) on disk in case we have to reload later

A>DDT SORT.COM, restart DDT with saved memory image

16K DDT VER 1.0

NEXT PC

0200 0100 "COM" file always starts with address 100H

-G, run the program from PC=100H

*0110 programmed stop (RST7) encountered

-D148

0148 05 00 07 00 14 00 1E 00 ← data properly sorted

0150 32 00 64 00 64 00 2C 01 E8 03 01 80 00 00 00 00 2.D.D., ...

0160 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00

0170 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00

-G, return to CP/M

ED SORT.ASM, make changes to original program

ch-2
*N, 0 (2) 0TT, find next "0"

MVI M, 0 ; I = 0

*-2 up one line in text

LXI H, 1 ; ADDRESS INDEX

*-2 up another line

MVI M, 1 ; SET TO 1 FOR FIRST ITERATION

*KT, kill line and type next line

LXI H, 1 ; ADDRESS INDEX

*I, insert new line

MVI M, 0 ; ZERO SW

*T,

LXI H, 1 ; ADDRESS INDEX

*NJNC (2) 0T,

JNC *T,

CONT ; CONTINUE IF I <= (N-2)

*-2 DIC (2) 0LT,

JC CONT ; CONTINUE IF I <= (N-2)

*E,

source from disk A

hex to disk A

ASM SORT.AAZ, skip prn file

CP/M ASSEMBLER - VER 1.0

0150 next address to assemble

003H USE FACTOR

END OF ASSEMBLY

DDT SORT.HEX, test program changes

16K DDT VER 1.0

NEXT PC

0150 0000

-G100,

*0110

-D140,

0140 05 00 07 00 14 00 1E 00 data sorted

0150 32 00 64 00 64 00 2C 01 EB 03 01 80 00 00 00 00 00 2 D D

0160 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00

- abort with rubout

-GO, return to CP/M - program checks OK.



Post Office Box 579, Pacific Grove, California 93950, (408) 649-3896

CP/M DYNAMIC DEBUGGING TOOL (DDT)
USER'S GUIDE

COPYRIGHT (c) 1976, 1978

DIGITAL RESEARCH

Table of Contents

Section	Page
I. INTRODUCTION	1
II. DDT COMMANDS	3
1. The A (Assemble) Command	3
2. The D (Display) Command	4
3. The F (Fill) Command	4
4. The G (Go) Command	4
5. The I (Input) Command	5
6. The L (List) Command	6
7. The M (Move) Command	6
8. The R (Read) Command	6
9. The S (Set) Command	7
10. The T (Trace) Command	7
11. The U (Untrace) Command	8
12. The X (Examine) Command	8
III. IMPLEMENTATION NOTES	9
IV. AN EXAMPLE	10

CP/M Dynamic Debugging Tool (DDT)

User's Guide

I. Introduction.

The DDT program allows dynamic interactive testing and debugging of programs generated in the CP/M environment. The debugger is initiated by typing one of the following commands at the CP/M Console Command level

```
DDT
DDT filename.HEX
DDT filename.COM
```

where "filename" is the name of the program to be loaded and tested. In both cases, the DDT program is brought into main memory in the place of the Console Command Processor (refer to the CP/M Interface Guide for standard memory organization), and thus resides directly below the Basic Disk Operating System portion of CP/M. The BDOS starting address, which is located in the address field of the JMP instruction at location 5H, is altered to reflect the reduced Transient Program Area size.

The second and third forms of the DDT command shown above perform the same actions as the first, except there is a subsequent automatic load of the specified HEX or COM file. The action is identical to the sequence of commands

```
DDT
Ifilename.HEX or Ifilename.COM
R
```

where the I and R commands set up and read the specified program to test (see the explanation of the I and R commands below for exact details).

Upon initiation, DDT prints a sign-on message in the format

```
nnK DDT-s VER m.m
```

where nn is the memory size (which must match the CP/M system being used), s is the hardware system which is assumed, corresponding to the codes

```
D   - Digital Research standard version
M   - MDS version
I   - IMSAI standard version
O   - Omron systems
S   - Digital Systems standard version
```

and m.m is the revision number.

Following the sign on message, DDT prompts the operator with the character "-" and waits for input commands from the console. The operator can type any of several single character commands, terminated by a carriage return to execute the command. Each line of input can be line-edited using the standard CP/M controls

rubout	remove the last character typed
ctl-U	remove the entire line, ready for re-typing
ctl-C	system reboot

Any command can be up to 32 characters in length (an automatic carriage return is inserted as the 33rd character), where the first character determines the command type

A	enter assembly language mnemonics with operands
D	display memory in hexadecimal and ASCII
F	fill memory with constant data
G	begin execution with optional breakpoints
I	set up a standard input file control block
L	list memory using assembler mnemonics
M	move a memory segment from source to destination
R	read program for subsequent testing
S	substitute memory values
T	trace program execution
U	untraced program monitoring
X	examine and optionally alter the CPU state

The command character, in some cases, is followed by zero, one, two, or three hexadecimal values which are separated by commas or single blank characters. All DDT numeric output is in hexadecimal form. In all cases, the commands are not executed until the carriage return is typed at the end of the command.

At any point in the debug run, the operator can stop execution of DDT using either a ctl-C or G0 (jmp to location 0000H), and save the current memory image using a SAVE command of the form

SAVE n filename.COM

where n is the number of pages (256 byte blocks) to be saved on disk. The number of blocks can be determined by taking the high order byte of the top load address and converting this number to decimal. For example, if the highest address in the Transient Program Area is 1234H then the number of pages is 12H, or 18 in decimal. Thus the operator could type a ctl-C during the debug run, returning to the Console Processor level, followed by

SAVE 18 X.COM

The memory image is saved as X.COM on the diskette, and can be directly executed by simply typing the name X. If further testing is required, the memory image can be recalled by typing

DDT X.COM

which reloads previously saved program from location 100H through page 18 (12FFH). The machine state is not a part of the COM file, and thus the program must be restarted from the beginning in order to properly test it.

II. DDT COMMANDS.

The individual commands are given below in some detail. In each case, the operator must wait for the prompt character (-) before entering the command. If control is passed to a program under test, and the program has not reached a breakpoint, control can be returned to DDT by executing a RST 7 from the front panel (note that the rubout key should be used instead if the program is executing a T or U command). In the explanation of each command, the command letter is shown in some cases with numbers separated by commas, where the numbers are represented by lower case letters. These numbers are always assumed to be in a hexadecimal radix, and from one to four digits in length (longer numbers will be automatically truncated on the right).

Many of the commands operate upon a "CPU state" which corresponds to the program under test. The CPU state holds the registers of the program being debugged, and initially contains zeroes for all registers and flags except for the program counter (P) and stack pointer (S), which default to 100H. The program counter is subsequently set to the starting address given in the last record of a HEX file if a file of this form is loaded (see the I and R commands).

1. The A (Assemble) Command. DDT allows inline assembly language to be inserted into the current memory image using the A command which takes the form

As

where s is the hexadecimal starting address for the inline assembly. DDT prompts the console with the address of the next instruction to fill, and reads the console, looking for assembly language mnemonics (see the Intel 8080 Assembly Language Reference Card for a list of mnemonics), followed by register references and operands in absolute hexadecimal form. Each successive load address is printed before reading the console. The A command terminates when the first empty line is input from the console.

Upon completion of assembly language input, the operator can review the memory segment using the DDT disassembler (see the L command).

Note that the assembler/disassembler portion of DDT can be overlaid by the transient program being tested, in which case the DDT program responds with an error condition when the A and L commands are used (refer to Section IV).

2. The D (Display) Command. The D command allows the operator to view the contents of memory in hexadecimal and ASCII formats. The forms are

D
Ds
Ds,f

In the first case, memory is displayed from the current display address (initially 100H), and continues for 16 display lines. Each display line takes the form shown below

aaaa bb bb bb bb bb bb bb bb bb bb bb bb bb bb cccccccccccccccc

where aaaa is the display address in hexadecimal, and bb represents data present in memory starting at aaaa. The ASCII characters starting at aaaa are given to the right (represented by the sequence of c's), where non-graphic characters are printed as a period (.) symbol. Note that both upper and lower case alphabets are displayed, and thus will appear as upper case symbols on a console device that supports only upper case. Each display line gives the values of 16 bytes of data, except that the first line displayed is truncated so that the next line begins at an address which is a multiple of 16.

The second form of the D command shown above is similar to the first, except that the display address is first set to address s. The third form causes the display to continue from address s through address f. In all cases, the display address is set to the first address not displayed in this command, so that a continuing display can be accomplished by issuing successive D commands with no explicit addresses.

Excessively long displays can be aborted by pushing the rubout key.

3. The F (Fill) Command. The F command takes the form

Fs,f,c

where s is the starting address, f is the final address, and c is a hexadecimal byte constant. The effect is as follows: DDT stores the constant c at address s, increments the value of s and tests against f. If s exceeds f then the operation terminates, otherwise the operation is repeated. Thus, the fill command can be used to set a memory block to a specific constant value.

4. The G (Go) Command. Program execution is started using the G command, with up to two optional breakpoint addresses. The G command takes one of the forms

G
Gs
Gs,b

Gs,b,c
G,b
G,b,c

The first form starts execution of the program under test at the current value of the program counter in the current machine state, with no breakpoints set (the only way to regain control in DDT is through a RST 7 execution). The current program counter can be viewed by typing an X or XP command. The second form is similar to the first except that the program counter in the current machine state is set to address s before execution begins. The third form is the same as the second, except that program execution stops when address b is encountered (b must be in the area of the program under test). The instruction at location b is not executed when the breakpoint is encountered. The fourth form is identical to the third, except that two breakpoints are specified, one at b and the other at c. Encountering either breakpoint causes execution to stop, and both breakpoints are subsequently cleared. The last two forms take the program counter from the current machine state, and set one and two breakpoints, respectively.

Execution continues from the starting address in real-time to the next breakpoint. That is, there is no intervention between the starting address and the break address by DDT. Thus, if the program under test does not reach a breakpoint, control cannot return to DDT without executing a RST 7 instruction. Upon encountering a breakpoint, DDT stops execution and types

*d

where d is the stop address. The machine state can be examined at this point using the X (Examine) command. The operator must specify breakpoints which differ from the program counter address at the beginning of the G command. Thus, if the current program counter is 1234H, then the commands

G,1234
and
G400,400

both produce an immediate breakpoint, without executing any instructions whatsoever.

5. The I (Input) Command. The I command allows the operator to insert a file name into the default file control block at 5CH (the file control block created by CP/M for transient programs is placed at this location; see the CP/M Interface Guide). The default FCB can be used by the program under test as if it had been passed by the CP/M Console Processor. Note that this file name is also used by DDT for reading additional HEX and COM files. The form of the I command is

Ifilename
or

Ifilename.filetype

If the second form is used, and the filetype is either HEX or COM, then subsequent R commands can be used to read the pure binary or hex format machine code (see the R command for further details).

6. The L (List) Command. The L command is used to list assembly language mnemonics in a particular program region. The forms are

L
Ls
Ls,f

The first command lists twelve lines of disassembled machine code from the current list address. The second form sets the list address to s, and then lists twelve lines of code. The last form lists disassembled code from s through address f. In all three cases, the list address is set to the next unlisted location in preparation for a subsequent L command. Upon encountering an execution breakpoint, the list address is set to the current value of the program counter (see the G and T commands). Again, long typeouts can be aborted using the rubout key during the list process.

7. The M (Move) Command. The M command allows block movement of program or data areas from one location to another in memory. The form is

Ms,f,d

where s is the start address of the move, f is the final address of the move, and d is the destination address. Data is first moved from s to d, and both addresses are incremented. If s exceeds f then the move operation stops, otherwise the move operation is repeated.

8. The R (Read) Command. The R command is used in conjunction with the I command to read COM and HEX files from the diskette into the transient program area in preparation for the debug run. The forms are

R
Rb

where b is an optional bias address which is added to each program or data address as it is loaded. The load operation must not overwrite any of the system parameters from 000H through 0FFH (i.e., the first page of memory). If b is omitted, then b=0000 is assumed. The R command requires a previous I command, specifying the name of a HEX or COM file. The load address for each record is obtained from each individual HEX record, while an assumed load address of 100H is taken for COM files. Note that any number of R commands can be issued following the I command to re-read the program under test,

assuming the tested program does not destroy the default area at 5CH. Further, any file specified with the filetype "COM" is assumed to contain machine code in pure binary form (created with the LOAD or SAVE command), and all others are assumed to contain machine code in Intel hex format (produced, for example, with the ASM command).

Recall that the command

```
DDT filename.filetype
```

which initiates the DDT program is equivalent to the commands

```
DDT
-Ifilename.filetype
-R
```

Whenever the R command is issued, DDT responds with either the error indicator "?" (file cannot be opened, or a checksum error occurred in a HEX file), or with a load message taking the form

```
NEXT PC
nnnn pppp
```

where nnnn is the next address following the loaded program, and pppp is the assumed program counter (100H for COM files, or taken from the last record if a HEX file is specified).

9. The S (Set) Command. The S command allows memory locations to be examined and optionally altered. The form of the command is

```
Ss
```

where s is the hexadecimal starting address for examination and alteration of memory. DDT responds with a numeric prompt, giving the memory location, along with the data currently held in the memory location. If the operator types a carriage return, then the data is not altered. If a byte value is typed, then the value is stored at the prompted address. In either case, DDT continues to prompt with successive addresses and values until either a period (.) is typed by the operator, or an invalid input value is detected.

10. The T (Trace) Command. The T command allows selective tracing of program execution for 1 to 65535 program steps. The forms are

```
T
Tn
```

In the first case, the CPU state is displayed, and the next program step is executed. The program terminates immediately, with the termination address

displayed as

*hhhh

where hhhh is the next address to execute. The display address (used in the D command) is set to the value of H and L, and the list address (used in the L command) is set to hhhh. The CPU state at program termination can then be examined using the X command.

The second form of the T command is similar to the first, except that execution is traced for n steps (n is a hexadecimal value) before a program breakpoint is occurs. A breakpoint can be forced in the trace mode by typing a rubout character. The CPU state is displayed before each program step is taken in trace mode. The format of the display is the same as described in the X command.

Note that program tracing is discontinued at the interface to CP/M, and resumes after return from CP/M to the program under test. Thus, CP/M functions which access I/O devices, such as the diskette drive, run in real-time, avoiding I/O timing problems. Programs running in trace mode execute approximately 500 times slower than real time since DDT gets control after each user instruction is executed. Interrupt processing routines can be traced, but it must be noted that commands which use the breakpoint facility (G, T, and U) accomplish the break using a RST 7 instruction, which means that the tested program cannot use this interrupt location. Further, the trace mode always runs the tested program with interrupts enabled, which may cause problems if asynchronous interrupts are received during tracing.

Note also that the operator should use the rubout key to get control back to DDT during trace, rather than executing a RST 7, in order to ensure that the trace for the current instruction is completed before interruption.

11. The U (Untrace) Command. The U command is identical to the T command except that intermediate program steps are not displayed. The untrace mode allows from 1 to 65535 (0FFFFH) steps to be executed in monitored mode, and is used principally to retain control of an executing program while it reaches steady state conditions. All conditions of the T command apply to the U command.

12. The X (Examine) Command. The X command allows selective display and alteration of the current CPU state for the program under test. The forms are

X
Xr

where r is one of the 8080 CPU registers

C	Carry Flag	(0/1)
Z	Zero Flag	(0/1)

M	Minus Flag	(0/1)
E	Even Parity Flag	(0/1)
I	Interdigit Carry	(0/1)
A	Accumulator	(0-FF)
B	BC register pair	(0-FFFF)
D	DE register pair	(0-FFFF)
H	HL register pair	(0-FFFF)
S	Stack Pointer	(0-FFFF)
P	Program Counter	(0-FFFF)

In the first case, the CPU register state is displayed in the format

CfZfMfEfIf A=bb B=dddd D=dddd H=dddd S=dddd P=dddd inst

where f is a 0 or 1 flag value, bb is a byte value, and dddd is a double byte quantity corresponding to the register pair. The "inst" field contains the disassembled instruction which occurs at the location addressed by the CPU state's program counter.

The second form allows display and optional alteration of register values, where r is one of the registers given above (C, Z, M, E, I, A, B, D, H, S, or P). In each case, the flag or register value is first displayed at the console. The DDT program then accepts input from the console. If a carriage return is typed, then the flag or register value is not altered. If a value in the proper range is typed, then the flag or register value is altered. Note that BC, DE, and HL are displayed as register pairs. Thus, the operator types the entire register pair when B, C, or the BC pair is altered.

III. IMPLEMENTATION NOTES.

The organization of DDT allows certain non-essential portions to be overlayed in order to gain a larger transient program area for debugging large programs. The DDT program consists of two parts: the DDT nucleus and the assembler/disassembler module. The DDT nucleus is loaded over the Console Command Processor, and, although loaded with the DDT nucleus, the assembler/disassembler is overlayable unless used to assemble or disassemble.

In particular, the BDOS address at location 6H (address field of the JMP instruction at location 5H) is modified by DDT to address the base location of the DDT nucleus which, in turn, contains a JMP instruction to the BDOS. Thus, programs which use this address field to size memory see the logical end of memory at the base of the DDT nucleus rather than the base of the BDOS.

The assembler/disassembler module resides directly below the DDT nucleus in the transient program area. If the A, L, T, or X commands are used during the debugging process then the DDT program again alters the address field at 6H to include this module, thus further reducing the logical end of memory. If a program loads beyond the beginning of the assembler/disassembler module, the A and L commands are lost (their use produces a "?" in response), and the

trace and display (T and X) commands list the "inst" field of the display in hexadecimal, rather than as a decoded instruction.

IV. AN EXAMPLE.

The following example shows an edit, assemble, and debug for a simple program which reads a set of data values and determines the largest value in the set. The largest value is taken from the vector, and stored into "LARGE" at the termination of the program

```

ED SCAN.ASM
*1 1-1 ORG 100H ; START OF TRANSIENT AREA
    MVI B,LEN ; LENGTH OF VECTOR TO SCAN
    MVI C,0 ; LARGEST-EST VALUE SO FAR
LOOP: P_0_0_L LXI H,VECT ; BASE OF VECTOR
    MOV A,M ; GET VALUE
    SUB C ; LARGER VALUE IN C?
    JNC NFOUND ; JUMP IF LARGER VALUE NOT FOUND
    ; NEW LARGEST VALUE, STORE IT TO C
    MOV C,A
NFOUND: INX H ; TO NEXT ELEMENT
    DCR B ; MORE TO SCAN?
    JNZ LOOP ; FOR ANOTHER
;
; END OF SCAN, STORE C
    MOV A,C ; GET LARGEST VALUE
    STA LARGE
    JMP 0 ; REBOOT
;
; TEST DATA
VECT: DB 2,0,4,3,5,6,1,5
LEN EQU $-VECT ; LENGTH
LARGE DS 1 ; LARGEST VALUE ON EXIT
END
;
; *BOP
    ORG 100H ; START OF TRANSIENT AREA
    MVI B,LEN ; LENGTH OF VECTOR TO SCAN
    MVI C,0 ; LARGEST VALUE SO FAR
    LXI H,VECT ; BASE OF VECTOR
LOOP: MOV A,M ; GET VALUE
    SUB C ; LARGER VALUE IN C?
    JNC NFOUND ; JUMP IF LARGER VALUE NOT FOUND
    ; NEW LARGEST VALUE, STORE IT TO C
    MOV C,A
NFOUND: INX H ; TO NEXT ELEMENT
    DCR B ; MORE TO SCAN?
    JNZ LOOP ; FOR ANOTHER
;

```

Create Source
Program - underlined
characters typed
by programmer.
";" represents carriage
return.

```

END OF SCAN, STORE C
MOV     A,C      ;GET LARGEST VALUE
STA     LARGE
JMP     0        ;REBOOT

TEST DATA
VECT:   DB      2,0,4,3,5,6,1,5
LEN     EQU     $-VECT ;LENGTH
LARGE:  DS      1      ;LARGEST VALUE ON EXIT
END

```

*E ← End of Edit

ASM SCAN Start Assembler
 CP/M ASSEMBLER - VER 1.0

0122
 002H USE FACTOR
 END OF ASSEMBLY

Assembly Complete - Look at Program Listing

TYPE SCAN.PRN

Code Address	Machine Code	Source Program
0100		ORG 100H ;START OF TRANSIENT AREA
0100 0600		MVI B,LEN ;LENGTH OF VECTOR TO SCAN
0102 0E00		MVI C,0 ;LARGEST VALUE SO FAR
0104 211901		LXI H,VECT ;BASE OF VECTOR
0107 7E	LOOP:	MOV A,M ;GET VALUE
0108 91		SUB C ;LARGER VALUE IN C?
0109 D20D01		JNC NFOUND ;JUMP IF LARGER VALUE NOT FOUND
		; NEW LARGEST VALUE, STORE IT TO C
010C 4F		MOV C,A
010D 23	NFOUND:	INX H ;TO NEXT ELEMENT
010E 05		DCR B ;MORE TO SCAN?
010F C20701		JNZ LOOP ;FOR ANOTHER
		; END OF SCAN, STORE C
0112 79		MOV A,C ;GET LARGEST VALUE
0113 322101		STA LARGE
0116 C30000		JMP 0 ;REBOOT
		TEST DATA
0119 0200040305	VECT:	DB 2,0,4,3,5,6,1,5
0008 =	LEN	EQU \$-VECT ;LENGTH
0121	LARGE:	DS 1 ;LARGEST VALUE ON EXIT
0122		END

Code/data listing truncated
 Value of Equate

A>

DDT SCAN.HEX

Start Debugger using hex format machine code

16K DDT VER 1.0

NEXT PC

0121 0000

-X, last load address + 1

0020M0E010 A=00 B=0000 D=0000 H=0000 S=0100 P=0000 OUT 7F PC=0

-XP,

Examine registers before debug run

P=0000 100, Change PC to 100

-X, Look at registers again

PC changed.

0020M0E010 A=00 B=0000 D=0000 H=0000 S=0100 P=0100 MVI B,00

-L100,

Next instruction
to execute at PC=100

0100 MVI B,00
0102 MVI C,00
0104 LXI H,0119
0107 MOV A,M
0108 SUB C
0109 JNC 010D
010C MOV C,A
010D INX H
010E DCR B
010F JNZ 0107
0112 MOV A,C

Disassembled Machine
Code at 100H
(See Source Listing
for comparison)

-L,

0113 STA 0121
0116 JMP 0000
0119 STAX B
011A NOP
011B INR B
011C INX B
011D DCR B
011E MVI B,01
0120 DCR B
0121 LXI D,2200
0124 LXI H,0200

A little more
machine code
(note that Program
ends at location 116
with a JMP to 0000)

-A116, enter inline assembly mode to change the JMP to 0000 into a RST 7, which
will cause the program under test to return to DDT if 116H
is ever executed.

0116 RST 7,

0117, (single carriage return stops assemble mode)

-L113, List Code at 113H to check that RST 7 was properly inserted

0113 STA 0121

0116 RST 07

In Place of JMP

```

0117  NOP
0118  NOP
0119  STAX B
011A  NOP
011B  INR B
011C  INX B

```

-X, Look at registers

```

00Z0M0E010 A=00 B=0000 D=0000 H=0000 S=0100 P=0100 MVI B,00

```

-T, Execute Program for one step. initial CPU state, before $\downarrow$ is executed

```

00Z0M0E010 A=00 B=0000 D=0000 H=0000 S=0100 P=0100 MVI B,00*0102

```

-T, Trace one step again (note 08H in B) automatic breakpoint $\rightarrow$

```

00Z0M0E010 A=00 B=0800 D=0000 H=0000 S=0100 P=0102 MVI C,00*0104

```

-T, Trace again (Register C is cleared)

```

00Z0M0E010 A=00 B=0800 D=0000 H=0000 S=0100 P=0104 LXI H,0119*0107

```

-T3, Trace three steps

```

00Z0M0E010 A=00 B=0800 D=0000 H=0119 S=0100 P=0107 MOV A,M

```

```

00Z0M0E010 A=02 B=0800 D=0000 H=0119 S=0100 P=0108 SUB C

```

```

00Z0M0E011 A=02 B=0800 D=0000 H=0119 S=0100 P=0109 JNC 010D*010D

```

-D119, Display memory starting at 119H.

automatic breakpoint at 10DH

Address	02	00	04	03	05	06	01	Program data
0119	02	00	04	03	05	06	01	
0120	05	11	00	22	21	00	02	7E EB 77 13 23 EB 0B 78 B1 " ! . W. #. (X)
0130	02	27	01	03	03	29	00	00 00 00 00 00 00 00 00
0140	00	00	00	00	00	00	00	00 00 00 00 00 00 00 00
0150	00	00	00	00	00	00	00	00 00 00 00 00 00 00 00
0160	00	00	00	00	00	00	00	00 00 00 00 00 00 00 00
0170	00	00	00	00	00	00	00	00 00 00 00 00 00 00 00
0180	00	00	00	00	00	00	00	00 00 00 00 00 00 00 00
0190	00	00	00	00	00	00	00	00 00 00 00 00 00 00 00
01A0	00	00	00	00	00	00	00	00 00 00 00 00 00 00 00
01B0	00	00	00	00	00	00	00	00 00 00 00 00 00 00 00
01C0	00	00	00	00	00	00	00	00 00 00 00 00 00 00 00

Lowercase x

Data is displayed in ASCII with a "0" in the position of non-graphic characters

-X, Current CPU state $\downarrow$

```

00Z0M0E011 A=02 B=0800 D=0000 H=0119 S=0100 P=010D INX H

```

-T5, Trace 5 steps from current CPU state

```

00Z0M0E011 A=02 B=0800 D=0000 H=0119 S=0100 P=010D INX H

```

```

00Z0M0E011 A=02 B=0800 D=0000 H=011A S=0100 P=010E DCR B

```

```

00Z0M0E011 A=02 B=0700 D=0000 H=011A S=0100 P=010F JNZ 0107

```

```

00Z0M0E011 A=02 B=0700 D=0000 H=011A S=0100 P=0107 MOV A,M

```

```

00Z0M0E011 A=00 B=0700 D=0000 H=011A S=0100 P=0108 SUB C*0109

```

-U5, Trace without listing intermediate states

```

00Z1M0E111 A=00 B=0700 D=0000 H=011A S=0100 P=0109 JNC 010D*0108

```

-X, CPU state at end of U5,

```

00Z0M0E111 A=04 B=0600 D=0000 H=011B S=0100 P=0108 SUB C

```



```

010C  MOV  C, A
010D  INX  H
010E  DCR  B
010F  JNZ  0107
0112  MOV  A, C

```

-X,

P=0100,

-T10, Trace to see how patched version operates

Data is moved from A to C

```

C0Z0M0E010 A=00 B=0000 D=0000 H=0000 S=0100 P=0100 MYI B,00
C0Z0M0E010 A=00 B=0000 D=0000 H=0000 S=0100 P=0102 MYI C,00
C0Z0M0E010 A=00 B=0000 D=0000 H=0000 S=0100 P=0104 LXI H,0119
C0Z0M0E010 A=00 B=0000 D=0000 H=0119 S=0100 P=0107 MOV A,M
C0Z0M0E010 A=02 B=0000 D=0000 H=0119 S=0100 P=0108 SUB C
C0Z0M0E011 A=02 B=0000 D=0000 H=0119 S=0100 P=0109 JC 010D
C0Z0M0E011 A=02 B=0000 D=0000 H=0119 S=0100 P=010C MOV C,A
C0Z0M0E011 A=02 B=0002 D=0000 H=0119 S=0100 P=010D INX H
C0Z0M0E011 A=02 B=0002 D=0000 H=011A S=0100 P=010E DCR B
C0Z0M0E011 A=02 B=0702 D=0000 H=011A S=0100 P=010F JNZ 0107
C0Z0M0E011 A=02 B=0702 D=0000 H=011A S=0100 P=0107 MOV A,M
C0Z0M0E011 A=00 B=0702 D=0000 H=011A S=0100 P=0108 SUB C
C1Z0M1E010 A=FE B=0702 D=0000 H=011A S=0100 P=0109 JC 010D
C1Z0M1E010 A=FE B=0702 D=0000 H=011A S=0100 P=010D INX H
C1Z0M1E010 A=FE B=0702 D=0000 H=011B S=0100 P=010E DCR B
C1Z0M0E111 A=FE B=0602 D=0000 H=011B S=0100 P=010F JNZ 0107*0107

```

-X,

breakpoint after 16 steps

```

C1Z0M0E111 A=FE B=0602 D=0000 H=011B S=0100 P=0107 MOV A,M

```

-G, 108, Run from current PC and breakpoint at 108H

*0108

-X,

next data item

```

C1Z0M0E111 A=04 B=0602 D=0000 H=011B S=0100 P=0108 SUB C

```

-T,

Single step for a few cycles

```

C1Z0M0E111 A=04 B=0602 D=0000 H=011B S=0100 P=0108 SUB C*0109

```

-T,

```

C0Z0M0E011 A=02 B=0602 D=0000 H=011B S=0100 P=0109 JC 010D*010C

```

-X,

```

C0Z0M0E011 A=02 B=0602 D=0000 H=011B S=0100 P=010C MOV C,A

```

-G, Run to completion

*0116

-X,

```

C0Z1M0E111 A=03 B=0003 D=0000 H=0121 S=0100 P=0116 RST 07

```

-S121, look at the value of "LARGE"

0121 03, Wrong Value!

0122 00,

0123 22,

0124 21,

0125 00,

0126 02,

0127 7E -,

End of the S Command

-L100,

```

0100 MVI B,08
0102 MVI C,00
0104 LXI H,0119
0107 MOV A,M
0108 SUB C
0109 JC 010D
010C MOV C,A
010D INX H
010E DCR B
010F JNZ 0107
0112 MOV A,C

```

Review the Code

-L,

```

0113 STA 0121
0116 RST 07
0117 NOP
0118 NOP
0119 STAX B
011A NOP
011B INR B
011C INX B
011D DCR B
011E MVI B,01
0120 DCR B

```

-XP,

P=0116 100, Reset the PC

-I, Single step, and watch data values

0021M0E111 A=03 B=0003 D=0000 H=0121 S=0100 P=0100 MVI B,08*0102

-I,

0021M0E111 A=03 B=0003 D=0000 H=0121 S=0100 P=0102 MVI C,00*0104

-I,

Count set
largest set

0021M0E111 A=03 B=0000 D=0000 H=0121 S=0100 P=0104 LXI H,0119*0107

-I,

base address of data set

0021M0E111 A=03 B=0000 D=0000 H=0119 S=0100 P=0107 MOV A,M*0108

$\overline{I_1}$, first data item brought to A
 0021M0E111 A=02 B=0000 D=0000 H=0119 S=0100 P=0108 SUB C*0109
 $\overline{I_1}$,
 0020M0E011 A=02 B=0000 D=0000 H=0119 S=0100 P=0109 JC 010D*010C
 $\overline{I_1}$,
 0020M0E011 A=02 B=0000 D=0000 H=0119 S=0100 P=010C MOV C,A*010D
 $\overline{I_1}$, first data item moved to C correctly
 0020M0E011 A=02 B=0002 D=0000 H=0119 S=0100 P=010D INX H*010E
 $\overline{I_1}$,
 0020M0E011 A=02 B=0002 D=0000 H=011A S=0100 P=010E DCR B*010F
 $\overline{I_1}$,
 0020M0E011 A=02 B=0702 D=0000 H=011A S=0100 P=010F JNZ 0107*0107
 $\overline{I_1}$,
 0020M0E011 A=02 B=0702 D=0000 H=011A S=0100 P=0107 MOV A,M*0108
 $\overline{I_1}$, second data item brought to A
 0020M0E011 A=00 B=0702 D=0000 H=011A S=0100 P=0108 SUB C*0109
 $\overline{I_1}$, subtract destroys data value which was loaded!!!
 0120M1E010 A=FE B=0702 D=0000 H=011A S=0100 P=0109 JC 010D*010D
 $\overline{I_1}$,
 0120M1E010 A=FE B=0702 D=0000 H=011A S=0100 P=010D INX H*010E
 $\overline{L100}$,
 0100 MVI B,08
 0102 MVI C,00
 0104 LXI H,0119
 0107 MOV A,M
 0108 SUB C ← This should have been a CMP so that register A would not be destroyed.
 0109 JC 010D
 010C MOV C,A
 010D INX H
 010E DCR B
 010F JNZ 0107
 0112 MOV A,C
 $\overline{A108}$,
 0108 CMP C, hot patch at 108H changes SUB to CMP
 0109,
 $\overline{00}$, stop DDT for SAVE

SAVE 1 SCAN.COM

Save memory image

A>DDT SCAN.COM

Restart DDT

16K DDT VER 1.0

NEXT PC

0200 0100

-XP

P=0100

-L116

0116 RST 07

0117 NOP

0118 NOP

0119 STAX B

011A NOP

- (rubout)

} Look at code to see if it was properly loaded
(long typeout aborted with rubout)

-G.116 Run from 100H to completion

*0116

-XC Look at Carry (accidental typo)

C1

-X Look at CPU state

C121M0E111 A=06 B=0006 D=0000 H=0121 S=0100 P=0116 RST 07

-S121 Look at "Large" - it appears to be correct.

0121 06

0122 00

0123 22

-G0 stop DDT

ED SCAN.ASM

Re-edit the source program, and make both changes

*NSUB

*0LT

SUB C
*SSUB ZCMP Z0LT
CMP C

CLL-2
;LARGER VALUE IN C?

;LARGER VALUE IN C?

*2

JNC NFOUND ;JUMP IF LARGER VALUE NOT FOUND

*SNC ZC Z0LT
JC NFOUND ;JUMP IF LARGER VALUE NOT FOUND

*E2

ASM SCAN.AAZ, Re-assemble, selecting source from disk A
 hex to disk A
 Print to Z (selects no print file)

0122
 002H USE FACTOR
 END OF ASSEMBLY

DDT SCAN.HEX, Re-run debugger to check changes

16K DDT VER 1.0
 NEXT PC
 0121 0000
 -L116,

0116 JMP 0000 check to ensure end is still at 116H
 0119 STAX B
 011A NOP
 011B INR B
 - (rubout)

-G100,116, Go from beginning with breakpoint at end

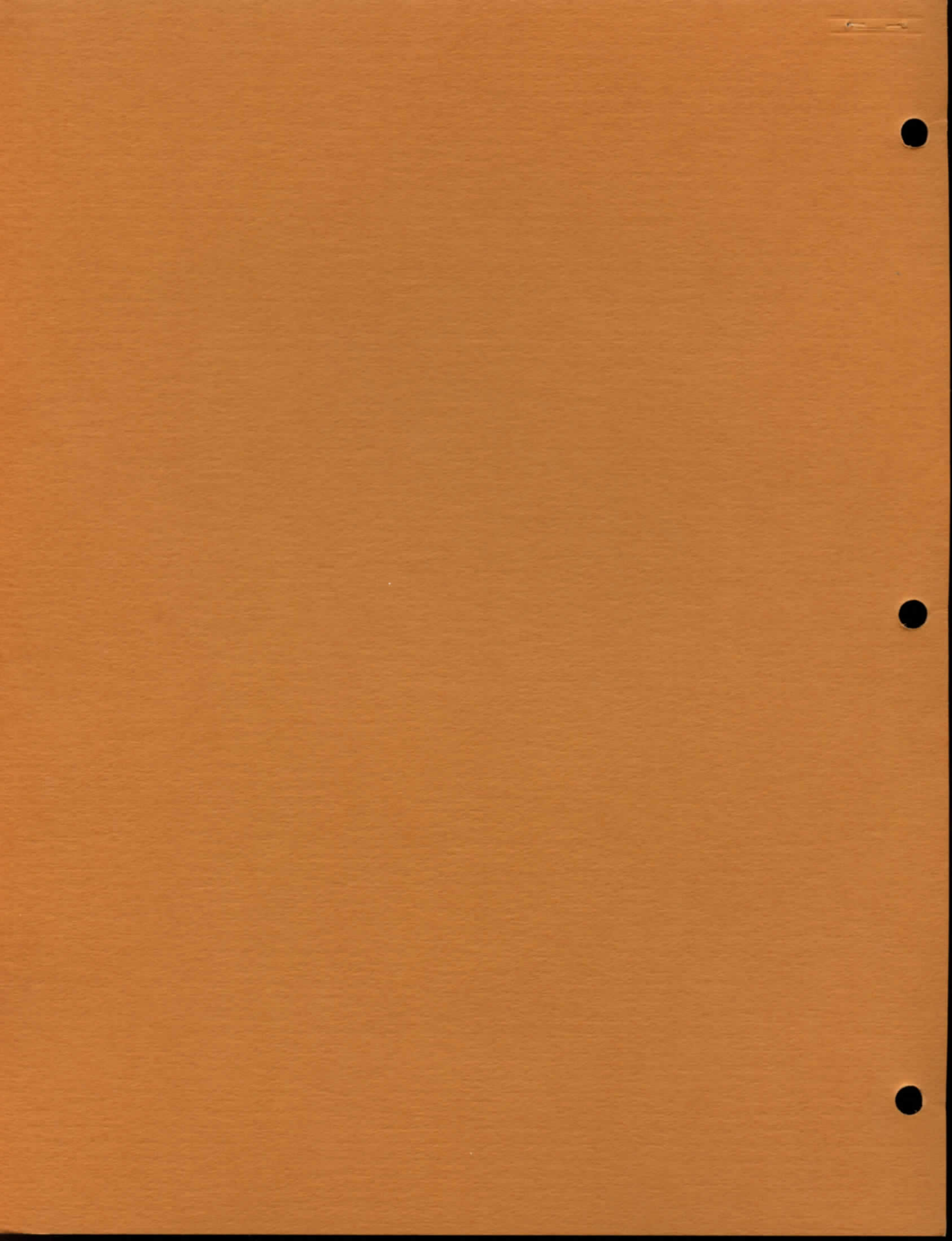
*0116 breakpoint reached

-D121, Look at "LARGE" correct value computed

0121 06 00 22 21 00 02 7E EB 77 13 23 EB 00 78 B1 ... "I...C.W.#...X."
 0130 C2 27 01 C3 03 29 00 00 00 00 00 00 00 00 00 ...
 0140 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ...

- (rubout) aborts long timeout

-G0, stop DDT, debug session complete



CP/M ON North Star DISK
Double Density Version

USERS NOTES

Lifeboat Associates
2248 Broadway
New York, N. Y. 10024

(212) 580-0082

Copyright (C) 1979

Revision 1.0
February 24, 1979

CP/M on North Star Double Density Disk Users Notes

INTRODUCTION TO THESE NOTES

This set of CP/M on North Star Disk Users Notes is intended to be used as supplementary material to the Digital Research manuals which accompany the system. It is not at all intended to serve as a stand alone guide to CP/M, but is aimed at assisting the new system owner in bringing the system up on their equipment as soon as possible. It is also aimed at informing the more experienced users about certain of the internal "hooks" and options which are specific to this particular CP/M system.

The Users Notes are divided into the following sections:

- Section I - STARTING OUT
- Section II - PROGRAMS PROVIDED
- Section III - SYSTEM REQUIREMENTS
- Section IV - SYSTEM LAYOUT
- Section V - SYSTEM OPTIONS
- Section VI - MISCELLANEOUS CP/M INFORMATION
- Section VII - BOOTING UP THE SYSTEM
- Section VIII - CONFIGURING YOUR I/O
- Section IX - CHANGING THE I/O CONFIGURATION
- APPENDIX A - Sample USER AREA Listings

Section I. STARTING OUT

Upon receiving your CP/M on North Star Disk package of diskette, and manuals, the first actions to take are:

- 1) Complete and return the Registration Card
- 2) Read the manuals
- 3) Back-up the diskette

Your CP/M on North Star is a serial numbered system licensed for your use on a single computer only. Unless the Registration Card is received by Lifeboat Associates, as is required under the terms of the license, no assistance can be offered with respect to queries you may wish answered. Remember that you are acquiring a complex and

CP/M on North Star Double Density Disk Users Notes

advanced piece of software which is under constant review and improvement both at Lifeboat Associates and Digital Research. We most strongly advise you to take advantage of the fact that this is a current and evolving system, and that you ensure that you are firmly plugged into the system for disseminating news and updates about CP/M.

Initially, at the minimum, read the Digital Research manual "An Introduction to CP/M Features and Facilities" in addition to this "CP/M on North Star Double Density Disk Users Notes". Thorough reading of these will make the process of bringing up the system much smoother. For further details, the user should also refer to the manuals "CP/M Interface Guide" and "CP/M System Alteration Guide".

We most strongly advise that your first act upon receiving the system is to put on a write protect tab and then make a working copy of the distribution diskette. This may be done with the North Star DOS using the "CD" command to copy the master disk onto your work disk. If you have a single drive system, use successive "RD" and "WR" commands until the ENTIRE disk is transferred.

It is very important that you NEVER WRITE on your original distribution diskette. Put it away in a safe place until (hopefully never) needed. It will provide a last resort back-up, regardless of what happens later, in the event of hardware or software malfunction. Use the disk you just created for normal work and make additional copies of it as needed. Remember to put the proper copyright notices on any disk you make as required by Digital Research.

Section II. PROGRAMS PROVIDED

These are programs supplied on your disk:

This first group of programs are standard CP/M distribution programs, the operation of which is described in the manual "Introduction to Features and Facilities".

MOVCPM.COM	- System relocater
ASM.COM	- Assembler
DDT.COM	- Debugger/Monitor
DUMP.COM	- Disk hex dump
ED.COM	- Editor
LOAD.COM	- Hex file loader
PIP.COM	- File interchanger
STAT.COM	- File parameters
SUBMIT.COM	- Batch processor
SYSGEN.COM	- System read/write
DUMP.ASM	- Sample .ASM program

This second group of programs are specially designed to be used with your North Star CP/M system. They all prompt the user at each step and may be run without fear of ruining something.

CP/M on North Star Double Density Disk Users Notes

LIST.COM	- Lists directory and number of records used for each program on the disk. Forms are: LIST, LIST A:, LIST B:
FORMAT.COM	- Blank disk formatter Will format a disk in selected drive to either single or double density.
DENSITY.COM	- Mixed density program Converts selected drive to either single or double density to run mixed density system.
COPY.COM	- Disk copy program The program will prompt you to select various options to copy the CP/M system, files, etc.
SAVEUSER.COM	- Quickly writes BIOS "patches" to disk by saving the MODE byte and USER AREA of the running system onto the disk.
CONFIG.COM	- Configures the USER AREA for commonly used I/O.
GENUSER.ASM	- Source listings of various USER AREAS from the CONFIG program.

Section III. SYSTEM REQUIREMENTS

In order to run CP/M on your computer you must have:

1. At least 24K of RAM starting at address zero.
2. A North Star double density disk system. A non-standard PROM is allowed.

Section IV. SYSTEM LAYOUT

The CP/M system you have received is designed to run in 24K of memory and is referred to as a "22K" system. The size of the CP/M system you can run is 2K less than available memory. The numbering is done in this way to conform with the addresses and standard CP/M system sizes as referred to in the various CP/M manuals. You will find more information on the memory usage of CP/M in the Alteration Guide. In general, when references are made to a 16K system with

CP/M on North Star Double Density Disk Users Notes

bias "b", the correct bias "b" to be used is 2800 hex for a 22K system.

The CP/M system resides on the disk on tracks 0 and 1. The directory and files start on track 2. The USER AREA which will contain your I/O drivers is a one page (256 byte) area on disk at disk address 17. The USER AREA is actually the second 256 bytes of this physical disk sector of 512 bytes in North Star convention.

The following table will list the locations of various CP/M modules in both the distribution 22K system and the "SYSGEN POSITION". The "SYSGEN POSITION" is the location of the CP/M system in memory when SYSGEN or MOVCPM is run. This position is the most useful for patching the system.

Module	22K System	SYSGEN
BOOT	4000 hex	900 hex
TPA	100	not appl.
CCP	4100	0A00
BDOS	4900	1200
BIOS	5600	1F00
MODEBYTE	5AFF	23FF
USER	5B00	2400
BUFFER	5C00	2500

Section V. SYSTEM OPTIONS

There are several user selectable options available in your CP/M system. These options are:

1. Read After Write.
2. Enable Interrupts after Disk Access.
3. Run AUTO after Warm Boot.
4. Run AUTO after Cold Boot.
5. Buffer Relocation
6. Density Select
7. Change Maximum Number of Tracks

The choice of the first four functions is controlled by the state of the bits in one byte referred to as the MODE byte. The MODE byte is located at the address immediately before the USER AREA and is at 5AFF hex in the standard 22K system (23FF hex in the SYSGEN POSITION). This address is always exactly 4FF hex above the BIOS or 4FC above the warm boot re-entry point of the BIOS. Since the warm boot re-entry point of the BIOS is always stored at location 0 in memory, programs may reference the MODE byte to manipulate it. A typical piece of code to perform this might be:

```
LHLD    1      ;Get warm boot address into HL
LXI     B,4FCH ;Use BC for the double add
```

CP/M on North Star Double Density Disk Users Notes

DAD	B	;Add HL and BC registers
		;to point to MODE byte
MOV	A,M	;Fetch MODE byte

The last three options are controlled by bytes located just prior to the MODE byte. The following paragraphs describe the controlled functions and indicate the trade-offs implied by selecting the use or suppression of each feature. Also explained are the position of each bit, the sense of the bit for invoking or disabling the function and the default value used in distribution systems. In general, the bit involved must be HIGH for the feature to be ACTIVE.

Option 1. READ AFTER WRITE

Use of Read After Write causes the system to read and verify the data on the disk after each sector is written. Any read error would result in repeated re-tries of the write and subsequent verification until the data is correctly written onto the disk. Use of this feature will cause no degradation in the speed of disk reading, but will slow disk writing by a factor of four times.

The position of this bit is value 40 hex. The distribution system is shipped with this bit LOW which causes this feature to be NOT active.

Option 2. ENABLE INTERRUPTS AFTER DISK ACCESS

The drivers of this system disable interrupts during a disk access, since the code is real time dependent. Otherwise, an interrupt during disk access would cause an error. Since the 8080 and 8085 processors are not able to read the status of the interrupt flag, the status cannot be restored to its original condition. This feature permits the user, either in general or under program control, to exit from disk accesses with interrupts either enabled or disabled.

The position of this bit is value 10 hex. The distribution system is shipped with this bit HIGH and thus this feature is active. That is, interrupts are ENABLED on exit from a disk access.

Option 3. RUN AUTO AFTER A WARM BOOT

Version 1.4 of CP/M includes a feature allowing execution of a command upon startup. This has applications in loading monitors into memory, initializing peripheral drivers, or causing the system to automatically enter an applications program for unskilled users. Your system has the command:

AUTO

set into it. This will cause a file AUTO.COM to be executed if the feature is invoked. The user must provide the program AUTO.COM or re-name a program to that name. The Warm Boot Auto feature will cause the command to be executed on each warm boot.

CP/M on North Star Double Density Disk Users Notes

The position of this bit is value 2. This distribution system is shipped with this bit set LOW and thus this feature is NOT active.

Option 4. RUN AUTO AFTER A COLD BOOT

As in the previous feature, except that the command is executed only on cold boots. The position of this bit is LOW and thus this feature is NOT active.

Option 5. BUFFER RELOCATION

Your CP/M system contains a 1K BUFFER from 5C00 to 5FFF hex in the 22K system as shipped (2500 to 28FF hex in the SYSGEN POSITION). When a COLD BOOT is done, the first 200 hex of this buffer is loaded along with the rest of the system from disk into memory. If you need more space in your USER AREA for your I/O routines, the location of this buffer may be moved, and the additional routines stored here. The location which stores the two byte address of this buffer is 5AFC in the 22K distribution system (23FC in the SYSGEN POSITION). This address would contain 5C00 hex in the 24K system as shipped. To change the buffer location, simply change the two byte address at location 23FC to your new buffer location and save the system with the re-located buffer on disk using SAVEUSER.

Option 6. DENSITY SELECT

Your system has been designed to read and write both single and double density North Star disks. A single byte in memory controls the density of all 4 drives. That byte is located at 5AFB hex in the distribution 22K system (23FB hex in the SYSGEN POSITION). Bits 1,3,5 and 7 control the density of drives A,B,C and D respectively. The density is single if the bit is set LOW, and double is set HIGH.

The distribution system is shipped with this byte equal to 0AA hex, which means all 4 drives are set to double density. These bits may be changed "on the fly" while the system is running or any specific mixture of densities may be set and permanently saved on the disk with SAVEUSER. You will then be able to run any of the CP/M programs on your mixed density system.

The program DENSITY.COM shipped with your system manipulates this byte to allow dual density systems to be conveniently run. You may want to, for example, set drive B to single density while leaving drive A at double density to transfer programs or do other CP/M functions. It is usually best to leave drive A in double density, as any CP/M program which exits with a warm boot, as many do, will crash because there cannot be a proper double density system (as opposed to files) on a single density disk.

Read this if you have a single drive system. This is the procedure to transfer programs from single to double density using a single drive. Two changes are necessary to do this. First, make sure your drive selector plug or switch on the rear of the circuit board

CP/M on North Star Double Density Disk Users Notes

is set to enable all three drives. This is the way it is shipped from the factory. Second, change one byte in your CP/M system located at 5AF5 hex (23F5 in SYSGEN) from a zero to an 01 hex. This patch may be saved if you wish with SAVEUSER. You may now run the DENSITY program and set drive B to single density. Do not worry that there is no drive B in your system. The fixes you have just made will make CP/M "think" that you are running a double density drive A and a single density drive B. Here is a sample dialog that would be used to transfer a file named TEST.ASM from a single to double density disk. The procedure is rather cumbersome because CP/M was not really intended to be run as a single drive system, even though it can be done.

A>	Log in with double density disk
DENSITY CR	CP/M prompt
A>	Run program, set B: to single density.
	CP/M prompt upon return
	Insert single density disk
B: CR	Login on "B:"
B>	CP/M prompt
DIR CR	Check directory of single density disk.
	Insert double density disk
A:DDT CR	Bring in DDT.COM from DD disk.
-	Insert single density disk
ITEST.ASM CR	DDT prompt
-	Get file into memory with DDT
R CR	DDT prompt
PC NEXT	using "I" and "R" commands.
1280 100	These next 3 lines typed by DDT
-	End and start address of program
	DDT prompt
S4 CR	Use "S" command to SET the byte
	at 4 hex from 1 to a 0.
0004 01	This allows re-boot to A:
0 CR	DDT responds
0005 00	You type to set byte
	DDT goes to next byte.
-	Type ". CR" to exit "S" mode.
	DDT prompt
	Insert double density disk
G0 CR	Do a warm boot
A>	CP/M prompt
SAVE 18 TEST.ASM CR	Save 18 pages of program

Note that the correct number of (256 byte) pages to SAVE is given by the hex address given by DDT. Take the first two digits under "PC", convert from hex to decimal, and SAVE that number of pages.

Option 7. CHANGE MAXIMUM NUMBER OF TRACKS

CP/M on North Star Double Density Disk Users Notes

The maximum number of tracks which can be accessed by the system is set by a byte in memory called NTRKS. NTRKS is located at 5AFA hex in the distribution 22K system (23FA hex in the SYSGEN POSITION). The distribution disk is shipped with NTRKS set to 35, which is the correct number for the Shugart drives shipped by North Star. If you have a drive with larger capacity, such as a Pertec with 40 track capacity, this byte may be changed. If you do so, you must also change a byte called DDMAXALL located at 5AEC hex in the distribution 22K system (23EC hex in the SYSGEN POSITION). DDMAXALL is currently set to 164 decimal (0A4 hex), which is one less than the maximum number of 1K file blocks which the CP/M system can access with a 35 track drive. This is calculated by multiplying 33 available file tracks (not including 2 system tracks) times 10 (512 byte) sectors per track divided by 2 sectors per (1K) block. For example, DDMAXALL would be changed from 164 decimal to 189 decimal for 40 track drives. This change will NOT affect single density. Both NTRKS and DDMAXALL may be saved with SAVEUSER. The FORMAT program is now set for 35 tracks. Look at the ASCII message at the beginning of the program with DDT to find a location to patch in more tracks.

Section VI. MISCELLANEOUS CP/M INFORMATION

1. You may get the error message LOAD ERROR when trying to run a long program under CP/M. This means the program you are attempting to run is too large for the current size CP/M system you are using. The cure for this is to use the MOVCPM program provided on your disk to generate a larger size system. Use STAT.COM to find the size of the program you wish to run. Then generate a CP/M system 7K larger. For example, if the program has a size of 29K as given by STAT, you must generate a 36K CP/M by using the command:

```
MOVCPM 36 CR
```

You may alternatively automatically generate the largest possible CP/M system that can fit into your available memory space by giving the command:

```
MOVCPM * CR
```

2. CP/M stores files on disk in a logically skewed order. That is, if a file is 5 sectors long, CP/M would not necessarily store it on sectors 1,2,3,4 and 5 in that order but might use a different sequence in order to make disk access times faster. This only applies to files, the system tracks (tracks 0 and 1) are in straight numerical order.

In double density CP/M each 512 byte physical sector on the disk is re-numbered for CP/M as four 128 byte sectors. For example, 512 byte sector 0 would be CP/M sectors 1,2,3 and 4. You will note in the table below that every four adjacent sectors are in numerical order, since they are actually one physical 512 byte sector. The sequence used in double density CP/M is:

```
(1,2,3,4),(21,22,23,24),(5,6,7,8),(25,26,27,28),(9,10,11,12)
(29,30,31,32),(13,14,15,16),(33,34,35,36),(17,18,19,20),(37,38,39,40)
```

CP/M on North Star Double Density Disk Users Notes

In single density CP/M on North Star each 256 byte physical sector on the disk is re-numbered for as two 128 byte sectors. The sequence for single density CP/M on North Star happens to come out in numerical order and is given by this table:

1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20

3. You must format raw diskettes before use with the program FORMAT.COM. This ensures that the disk has been properly initialized for CP/M. Just give the command "FORMAT CR" and the program will ask you which drive to use. The disk is ready for use after FORMAT runs. and whether to format in single or double density. Alternatively, you may use the North Star DOS "IN" command but if you do, the first time that disk is used by CP/M you must initialize the directory with the command: "ERA *.* CR". CP/M will ask "ALL FILES Y/N?" and your response should be "Y CR".

Section VII. BOOTING UP THE SYSTEM

The purpose of this section is to guide you in getting your CP/M system running for the first time. This is a two stage process. The first stage, covered in this section, is to bring the CP/M system into memory. The second stage, covered in the next section "CONFIGURING YOUR I/O", involves permanently customizing the system for your specific I/O. We suggest you read both sections as a unit several times to get a "feel" for the procedures. Then start right here, do this section first and then the next.

First make sure you have at least 24K of RAM memory starting at 0. Also MAKE SURE you are using a "working" disk which is a COPY of the distribution diskette. All tracks should have been copied. DO NOT USE THE ORIGINAL.

There are two methods of booting up. One is the DOS method which has been designed to be as easy and painless as possible. Use this method if possible because it will give you immediate console control of the system and is a quick and positive test of the system and your equipment. The other is the NORMAL method which you will always use when more familiar with the system. Although you do not gain immediate console control, this much more general method may always be used. Pick either the DOS or NORMAL method, go to the section on the method you have selected, do that procedure and then go to the next section on "CONFIGURING YOUR I/O". Throughout this text, CR in a command means a carriage return.

Method 1. DOS METHOD

This method may ONLY be used if your PROM is in the standard 0E800 hex location and your North Star DOS is in the standard 2000 hex location. Use the NORMAL method if either of these differ. This DOS method does it's I/O through your existing North Star DOS user area. Whatever you have previously patched into the area of North Star DOS between 2900 and 29FF hex will now be temporarily used by CP/M until a more permanent I/O is installed into the CP/M system itself. This is the procedure:

Step 1. Boot in your North Star DOS in the usual fashion.

Step 2. Insert the working copy CP/M diskette in drive A.

Step 3. Type the command: GO CPM CR

Step 4. Your system should come up with the sign-on message:

CP/M on North Star Disk
22K Version 1. 44
Copyright (C) 1979 Lifeboat Associates

A>

CP/M on North Star Double Density Disk Users Notes

Congratulations, your system is on the air. Please remember that your I/O is being done thru North Star DOS, which is right in the middle of where many CP/M programs run. This is only designed to be a TEMPORARY system until your own I/O drivers have been patched in.

In the meantime, you may use all the built-in functions such as DIR, TYPE, ERA, REN and SAVE. You may also use (ONLY) the COM programs DDT, LIST, STAT, DENSITY, SAVEUSER and CONFIG. Don't try to run any of the others, the system will crash. Now go to the section on "CONFIGURING YOUR I/O" to make the permanent I/O patches in your system.

Method 2. NORMAL METHOD

This method can be used with both standard and non-standard PROM or DOS addresses. The system makes any necessary adjustments when it is booted in. The NORMAL method works by using the PROM to load the CP/M system into the proper memory area. If the system has been previously configured it will simply sign-on and be ready to run. If your I/O has not been previously configured, a special program called CONFIG will be automatically loaded which may be used by you to customize the system with your specific I/O as explained in the section "CONFIGURING YOUR I/O". You may optionally pre-configure the system by transferring the I/O area from an existing single density CP/M disk as explained in the next paragraph. Do so now if you elect. Then go to NORMAL BOOT PROCEDURE.

USING EXISTING SINGLE DENSITY USER AREA

Do this only if you have been running single density CP/M on North Star. Otherwise, go directly to NORMAL BOOT PROCEDURE.

This procedure will allow you to transfer the USER AREA currently in your single density CP/M system to your new double density system. This method has the advantage that your system will immediately sign-on without going thru the configuration procedure in the next section. This is the procedure to use if your I/O is complex or not covered in the list of standard terminals which can be automatically configured by CONFIG. The existing single density CP/M from which you will take the USER AREA must be exactly 22K in system size. USER AREA size is limited to one page of 256 bytes in the double density system.

- Step 1. Boot in double density North Star DOS.
- Step 2. Insert new double density CP/M disk.
- Step 3. Read in 512 bytes of new disk using DOS command:
RD 17 100 1 CR
New USER AREA is now at 200 hex ready to modify.
- Step 4. Insert old single density CP/M disk.
- Step 5. Overlay above with old USER AREA using command:
RD 28 200 1 S CR
- Step 6. Re-insert new double density disk.
- Step 7. Write USER AREA back to double density disk
using DOS command:

WR 17 100 1 CR

NORMAL BOOT PROCEDURE

Step 1. Boot in your North Star DOS in the usual fashion. This ensures that your hardware has been properly initialized. If you can't do this, just go to the next step.

Step 2. Insert the (COPY of distribution diskette) working CP/M diskette in drive A.

Step 3. Run the computer from the entry point of your PROM which is normally 0E800 hex but may be another address if non-standard. If your North Star DOS is running, type the command:

JP E800 CR (or non-standard address).

If DOS is not running, use your monitor, front panel switches or whatever other facilities you have to jump to E800 hex (or non-standard address) to boot in the system.

Step 4. If you have a single density CP/M on North Star and have already modified the USER AREA on your double density by writing onto it the old single density USER AREA, (as explained in the previous paragraph) the system will now sign-on. The system is running and you have completed the boot-up procedure. Otherwise, go on to the next step.

Step 5. Your disk will be accessed and then stop. You will NOT get any message on the console. What happened was that the CP/M system was brought into memory and it then loaded and ran CONFIG. The CONFIG program is designed to come to a halt until you go thru the steps in the "CONFIGURING YOUR I/O" in the next section.

TO GENERATE NEW VERSION FROM 22K VERSION

A) BOOT 22K VERSION

B) RUN MOVCPM

A> MOVCPM 56 * ↓

SIZE

C) RUN SYSGEN

A> SYSGEN ↓

SOURCE DRIVE NAME (OR RETURN TO SKIP) ↓

DESTINATION DRIVE NAME (OR RETURN TO REBOOT) A ↓

IMPORTANT
SKIP

D) PATCH I/O AREA (USING ROM MONITOR)

AS ON PAGE 12

Section VIII. CONFIGURING YOUR I/O

Your CP/M on North Star system includes a program called CONFIG.COM which is designed to automatically patch the USER AREA of your system for a variety of different I/O devices. The USER AREA is defined as a one page (256 bytes) area in the CP/M system which begins with a table of jumps to your individual I/O routines. This table is accessed by other parts of the system. The actual I/O routines themselves are usually located in the remaining (approximately 240) bytes of the USER AREA. They may alternatively be located in PROM or a different part of RAM above the CP/M system. The I/O conventions in the Digital Research manual "System Alteration Guide" apply.

The CONFIG program contains a variety of prototype USER AREAS for different I/O devices in common use. The list of I/O devices included in CONFIG is given below. When run, CONFIG automatically inserts proper port numbers, status bit information and memory addresses into the prototype USER AREA, then moves the prototype to a temporary PATCH AREA located at 200 hex in memory for possible further modification by you, and finally moves the entire PATCH AREA to the USER AREA. You may accomplish this entire operation either in a single step if you have a standard configuration or in multiple steps if extensive patching is required.

If you used the NORMAL method of boot-up, the first part of CONFIG has been run and your system is halted and seems to be dead. Relax, it's just waiting for the next part, by doing a "soft halt" (jump to itself) at memory location 103 hex.

If you used the DOS method of boot-up, you must now get to this same stage by loading CONFIG. To do this, type:

DDT CONFIG.COM CR

DDT will sign on in the normal manner, and CONFIG will be in memory waiting to be run. This method has the advantage that you can use the facilities of DDT to help configure the system. Having DDT makes it easier to do the next step.

We are now ready to configure the system. Look in the table of configurations for the one pertaining to your I/O devices. Refer to APPENDIX A for the specific details and listings of each of these configurations.

LIST OF STANDARD TERMINALS

0. Altair 88-2SIO
1. Imsai SIO2
2. Altair SIO (Rev Non-Zero)
3. Altair SIO (Rev Zero)
4. Processor Technology 3P+3
5. Imsai MIO
6. Altair 88-4PIO
80. Compal 80
81. Processor Technology SOL

CP/M on North Star Double Density Disk Users Notes

- 82. Xitan SMB (Zapple)
- 83. North Star Horizon
- 84. Vector Graphic MZ

If your I/O device is listed, go to the heading on CONFIGURING FOR A STANDARD TERMINAL. If it is not listed, go to CONFIGURING FOR A NON-STANDARD TERMINAL.

CONFIGURING FOR A STANDARD TERMINAL

FIRST, use whatever facilities you have to deposit the configuration number of your choice in memory location 120 hex. If you used the DOS method and have the use of DDT, use the "S" command in DDT. Else, use your monitor, front panel switches or whatever it takes to get that number into location 120 hex.

SECOND, use whatever facilities you have to RUN from location 100 hex. If you have DDT, use the "G" command by typing:

G100 CR

Else, use your monitor or front panel switches.

You should get a sign-on message of the form:

```
CONFIG Version 1.3
Copyright (C) 1979 Lifeboat Associates
Your CP/M System is now configured.
Type SAVEUSER to permanently save on disk.
A>
```

At this point you should have a working system patched in memory with your I/O. It may be permanently written on the disk with the SAVEUSER program by typing:

SAVEUSER CR

The program will prompt you what to do. You should now test the system by doing a COLD BOOT. This brings in a totally fresh system from disk and is a good test of whether your patches were right. To do this, use your monitor, reset switch, front panel or whatever to boot in the system in exactly the same way you boot in the North Star DOS. It is best to cycle the computer AC power (after first removing all diskettes) before this test to wipe memory clean and perform a definite test of your device initialization.

CONFIGURING FOR A NON-STANDARD TERMINAL

At this point, the system is in memory, has loaded the CONFIG program, and is ready for the USER AREA at 5B00 hex to be patched with your individual console I/O. If you used the DOS method to boot-up, you have the use of DDT. Otherwise the system has loaded CONFIG and has come to a halt.

Using CONFIG for a non-standard terminal is a two step process. First, you must put a copy of your proposed USER AREA into a PATCH AREA located at 200 hex in memory. You may use CONFIG to do this, or patch in the code with your own monitor or front panel switches. Second, use CONFIG to move this PATCH AREA to the USER AREA where it

will actually run. You will find sample USER AREAs in the Appendix to serve as a guide.

After each step in running CONFIG, control will return to location 103 hex, which is normally a soft halt. That is, the computer will continuously loop at that address and seem to be stopped. If you have the use of DDT or another monitor, you may return to that monitor instead of stopping. To do this, use your monitor to store the entry address of your monitor at location 104 hex. For example, if you used the DOS method, use the "S" command of DDT to store a 0038 hex which is the re-entry address of DDT. You would store a 38 hex (which is the low byte of the address) at location 104 hex and a 0 (the high byte) at location 105 hex. This will cause CONFIG to return to DDT after you have run the program.

It is now necessary to decide if any of the standard configurations offer anything, so that modifying them is advantageous over starting from scratch. In particular, configuration numbers 0 thru 6 are standard teletype drivers which are used in many different systems. You may have to change the port numbers or the status bits but the subroutines are probably otherwise correct.

If you think one of the standard configurations is a good starting point, pick the one you want, and use your facilities to store the appropriate configuration number at 120 hex as explained above. Then use your facilities to jump to 106 hex which will produce the configuration you have selected and store it in the PATCH AREA starting at memory location 200 hex without making any change in your system USER AREA. The purpose of storing the configuration at the PATCH AREA is to allow you to conveniently modify it. Use your monitor to change the partially configured PATCH AREA to one you think might work.

If you don't think one of the standard configurations is a good starting point, use your monitor or front panel switches to patch in an entire prototype USER AREA. These patches must be inserted in the PATCH AREA. All address references in calls and jumps must be the ones desired in the final running system. CONFIG will not change any of the addresses. It will simply move the entire block of memory up to the USER AREA. Be sure to begin with the table of jumps in proper order. In appendix A you will find a listing of a sample user area using the teletype driver. Although this will not apply exactly to your case, it will serve as a guide for you to patch the system for your needs. The important thing in the beginning is to patch at least the drivers for CINIT, CONIN and CONOUT. The other jumps can be terminated with simple RETURNS until you wish to further patch your system.

Try out your patches by running CONFIG from memory location 109 hex. This will cause CONFIG to move the entire PATCH AREA at 200 hex to the USER AREA at 5B00 hex. CONFIG will then call the first jump in the USER AREA jump table which should jump to the code necessary to initialize your I/O devices. CONFIG will then attempt to do a WARM BOOT and sign on. If the system does sign on, you may save your patches on disk using SAVEUSER and then test your patches

by trying a COLD BOOT as explained in the previous section. If the system doesn't sign on, there is something wrong in your I/O routines. Since everything may have crashed, it is best to start fresh in this case. Keep at it until you see the CONFIG sign-on message on the screen.

Section IX. CHANGING THE I/O CONFIGURATION

The purpose of this section is to explain how to make changes in your I/O after the system is up and running. You may wish to implement a printer, have more than one console or many other variations on the basic console I/O that you began with. There are two methods of doing this. The SYSGEN PROCEDURE is a comprehensive method of making system changes. SAVEUSER is a quick and easy method of making short patches.

SYSGEN PROCEDURE

The following procedure will be referred to as the "SYSGEN PROCEDURE" for changing your system. This is an advanced procedure and should only be used after your system has been properly running and you are familiar with the system and the use of the ED.COM, ASM.COM, SYSGEN and DDT. It uses the facilities of CP/M to create an assembly language file containing your customized I/O. The existing system is then brought into memory at location 900 hex, which is referred to as the "SYSGEN POSITION". This is the standard and most convenient memory location for making changes in the CP/M system. The program resulting from your assembly language file is then inserted into the USER AREA of the system while in the SYSGEN POSITION and the new system is then written onto the disk.

The file GENUSER.ASM contains many different types of I/O. Changes may be made in this file to suit your own purposes and then implemented in your system. Pick out the most useful sections and modify this file using the CP/M editor ED.COM. Make sure any of the unused prototype USER AREAS in GENUSER.ASM are deleted. Also make sure your USER AREA has a proper ORG statement at BIOS+500 hex. Then assemble the changed file using the CP/M assembler ASM.COM. This will produce a file called GENUSER.HEX. Use SYSGEN.COM to GET a copy of the existing system into memory at the SYSGEN POSITION and save that as a file by the command "SAVE 36 CPM22.COM". Now make sure you are logged in on the same drive as the GENUSER.HEX file so that it can be inserted by DDT. Bring the existing system back to the SYSGEN POSITION by the command "DDT CPM22.COM". Now overlay the original USER AREA (2400 to 24FF hex) with your new GENUSER.HEX file using the two commands "IGENUSER.HEX" and "R<OFFSET>". OFFSET for a 22K system is 0C900 hex. For other size systems the correct OFFSET is given in the GENUSER.ASM file and represents the difference from where the USER AREA actually runs in memory (at 5B00 hex for a 22K system) to its location in the SYSGEN POSITION which is always 2400

CP/M on North Star Double Density Disk Users Notes

hex. Now you have a patched CP/M system in memory in the SYSGEN POSITION ready to be saved as a file by the command "SAVE 36 C24PAT.COM". Finally, this patched system may be put on your disk by the command "SYSGEN C24PAT.COM". SYSGEN will ask which drive you wish to put the system on and we suggest drive B: so you do not overwrite your old system until you are absolutely sure the new one works properly. Now the patched system is on B: ready to be inserted in drive A: and cold booted. It may be helpful for you to know that SYSGEN can always be run with a system in memory at the SYSGEN POSITION without disturbing that system.

SAVEUSER

If you wish to make quick and simple patches in your system without going thru the above procedure, we have included a file called SAVEUSER.COM. When SAVEUSER is run, it will write onto your drive A: disk the patched user area so that when you boot up in the future, these changes will be in your system. SAVEUSER can also be used to patch the system on your disk when the various SYSTEM OPTIONS are changed. SAVEUSER writes the two sectors containing the USER AREA and the previous sector from memory in the running system to the proper area on disk. You may therefore make a change in the running system, such as a MODE byte change, test it's operation and then permanently save that change.

APPENDIX A - SAMPLE USER AREA LISTING

```

;
;GENUSER                                MARCH 1, 1979
;
000A = PAGE 0
VERS EQU 10
;
;GENERAL USER AREAS FROM CONFIG
;
;THESE USER AREAS ARE DUPLICATES OF WHAT CONFIG WOULD PRODUCE.
;THEY MAY BE USED AS A BASIS FOR MODIFYING YOUR OWN ROUTINES.
;IN ACTUAL USE, YOU MUST PICK *** ONLY ONE ***
;
;*****
;*** SIMPLE TELETYPE DRIVER ***
;*****
;
0016 = MSIZE EQU 22 ;DECIMAL SIZE OF SYSTEM
5600 = BIOS EQU (MSIZE*1024)-200H ;STANDARD CP/M CONVENTION
5B00 = USER EQU BIOS+500H ;LOCATION TO USER AREA
C900 = OFFSET EQU 1F00H-BIOS ;TO SYSGEN IMAGE
0003 = IOBYT EQU 3 ;PAGE 0 LOCATION
;JUMPS MUST REMAIN HERE, IN SAME ORDER
0000 = ORG USER
;
; TEMPORARY CONSOLE DEVICE ASSIGNMENTS
; FILLED IN BY CONFIG WITH OTHER PORTS/MASKS
; THESE ARE VALID FOR CONFIGURATION 2
0000 = TTS EQU 0 ;CONSOLE STATUS PORT
0001 = TTI EQU 1 ;CONSOLE DATA INPUT PORT
0001 = TTO EQU TTI ;CONSOLE DATA OUTPUT PORT
0001 = RDA EQU 1 ;INPUT READY MASK
0000 = RDAMSK EQU 0 ;MASK TO CONVERT TO ACTIVE LOW
0080 = TBE EQU 80H ;OUTPUT READY MASK
0000 = TBEMSK EQU 0 ;MASK TO CONVERT TO ACTIVE LOW
;
;*** USER AREA JUMP TABLE ***
TTY0: ;BASE OF JUMP TABLE
5B00 C3595B JMP TTYCINIT ;COLD INITIALIZATION
5B03 C3205B JMP TTYST ;CONSOLE STATUS
5B06 C3305B JMP TTYIN ;CONSOLE INPUT
5B09 C3405B JMP TTYOUT ;CONSOLE OUTPUT
5B0C C3405B JMP TTYLIST ;LIST OUTPUT
5B0F C3405B JMP TTYPUNCH ;PUNCH OUTPUT
5B12 C3305B JMP TTYREADER ;READER INPUT
5B15 C3505B JMP TTYPRST ;PRINTER STATUS NOT IMP
5B18 000000 DB 0,0,0
5B1B 8000 DW TTYLEN ;LENGTH
5B1D 000000 DB 0,0,0
;
;*** CONSOLE STATUS CHECK ROUTINE ***
TTYST: ;RET ACC=0 IF NO KEY, ACC=0FFH IF KEY

```

CP/M on North Star Double Density Disk Users Notes

```

5B20 DB00          IN      TTS          ;GET STATUS
5B22 00            NOP
5B23 E601          ANI      RDA          ;AND WITH STATUS BITS
5B25 EE00          XRI      RDAMSK       ;MAKE ACTIVE LOW
5B27 3E00          MVI      A,0          ;NO KEY FLAG
5B29 C0            RNZ
5B2A 2F            CMA          ;NO KEY WAITING
5B2B C9            RET          ;KEY FLAG OFFH
5B2C 00000000      DB      0,0,0,0     ;KEY CHAR WAITING

```

;
;*** CONSOLE INPUT ROUTINE ***

TTYREADER: ;NOT IMPLEMENTED
TTYIN:

```

5B30 DB00          IN      TTS          ;GET STATUS
5B32 00            NOP
5B33 E601          ANI      RDA          ;AND WITH STATUS BITS
5B35 EE00          XRI      RDAMSK       ;MAKE ACTIVE LOW
5B37 C2305B        JNZ      TTYIN       ;LOOP UNTIL READY
5B3A DB01          IN      TTI          ;GET CHR FROM DATA
5B3C 00            NOP
5B3D E67F          ANI      7FH         ;STRIP PARITY
5B3F C9            RET          ;DONE

```

;
;*** CONSOLE OUTPUT ROUTINE ***

TTYLIST: ;NOT IMPLEMENTED
TTYPUNCH: ;NOT IMPLEMENTED
TTYOUT:

```

5B40 DB00          IN      TTS          ;GET STATUS
5B42 00            NOP
5B43 E680          ANI      TBE          ;AND WITH STATUS BITS
5B45 EE00          XRI      TBEMSK       ;MAKE ACTIVE LOW
5B47 C2405B        JNZ      TTYOUT       ;LOOP UNTIL READY
5B4A 79            MOV      A,C         ;INTO ACC FOR OUT
5B4B D301          OUT      TTO         ;OUTPUT IT
5B4D 00            NOP
5B4E C9            RET          ;DONE
5B4F 00            NOP

```

;
;*** PRINTER STATUS CHECK ***

TTYPRST:
5B50 000000 NOP ! NOP ! NOP
5B53 AF XRA A ;NO CHAR READY
5B54 C9 RET
5B55 00000000 NOP ! NOP ! NOP ! NOP

;
;*** COLD INIT ROUTINE ***

TTYCINIT: ;DEVICE INIT MOVED TO HERE

```

5B59 C9            RET
5B5A              DS      38

```

```

0080 =            TTYLEN EQU      $-TTY0
;*** END OF SIMPLE TELETYPE DRIVER ***
;
;
;

```

CP/M on North Star Double Density Disk Users Notes

```

;
;
;*** OTHER TELETYPE CONFIGURATIONS ***
;
;THE CORRECT PORT NUMBERS, STATUS BITS AND MASKS
;FOR EACH CONFIGURATION IS INSERTED INTO THE
;GENERAL ROUTINE ABOVE BY CONFIG.
;THEY ARE OBTAINED FROM THE FIRST LINE
;OF DATA BYTES FOR EACH CONFIGURATION.
;
;THE ORDER OF THE DATA BYTES IS:
;      TTS, TTI, TTO, RDA, RDAMSK, TBE, TBEMSK
;
;THEN, THE INDIVIDUAL INITIALIZATION ROUTINES
;ARE MOVED INTO PLACE AT TTYCINIT.
;
;THE CODE FROM HERE DOWN, ALTHOUGH ASSEMBLED IN
;THIS EXAMPLE, IS NOT MOVED INTO THE USER AREA BY CONFIG.
;ONLY THE SPECIFIC BYTES NEEDED ARE SELECTIVELY
;INSERTED INTO THE ABOVE GENERAL HANDLER BY CONFIG
;WHEN YOU SPECIFY A CONFIGURATION NUMBER.

```

```

;CONFIGURATION 0-- ALTAIR 88-2SIO

```

```

;      OR OTHER SIO USING MOTOROLA 6850 UART

```

```

5B80 1011110101CNFG0: DB      16,17,17,1,1,2,2
5B87 09                DB      9                ;INIT LENGTH
;INIT ROUTINE
5B88 3E03                MVI      A,3                ;RESET 6850
5B8A D310                OUT      16                ;PROGRAM FOR 8 BITS
5B8C 3E11                MVI      A,11H              ;2STOP,NOPARITY
5B8E D310                OUT      16                ;16 CLOCK
5B90 C9                RET                        ;DONE

```

```

;CONFIGURATION 1--IMSAI SIO2

```

```

;      OR OTHER SIO USING THE INTEL 8251 USART

```

```

5B91 0302020202CNFG1: DB      3,2,2,2,2,1,1
5B98 11                DB      17                ;INIT LENGTH
;INIT ROUTINE
5B99 3EAA                MVI      A,0AAH              ;DUMMY MODE BYTE
5B9B D303                OUT      3
5B9D 3E40                MVI      A,40H              ;RESET
5B9F D303                OUT      3
5BA1 3ECE                MVI      A,0CEH              ;MODE INSTRUCTION
5BA3 D303                OUT      3
5BA5 3E37                MVI      A,37H              ;COMMAND INSTRUCTION
5BA7 D303                OUT      3
5BA9 C9                RET                        ;DONE

```

```

;CONFIGURATION 2-- ALTAIR SIO A,B,C (NOT REV 0) - OLD MITS

```

```

;      OR OTHER UART TYPE SERIAL I/O BOARD

```

```

;      NOT REQUIRING INITIALIZATION

```

```

5BA0 0001010100CNFG2: DB      0,1,1,1,0,80H,0
5BB1 01                DB      1
;INIT ROUTINE
5BB2 C9                RET

```

CP/M on North Star Double Density Disk Users Notes

```

;
;CONFIGURATION 3 -- ALTAIR SIO A,B,C (REV 0)
5BB3 0000000101CNFG3: DB 0,0,0,1,1,20H,20H,20H,20H,2,2
5BBE 01 DB 1
INIT3: ;INIT ROUTINE
5BBF C9 RET ;DONE

;
;CONFIGURATION 4-- PROCESSOR TECHNOLOGY 3P+S SERIAL I/O
5BC0 0000000101CNFG4: DB 0,0,0,1,1,40H,40H,40H,40H,80H,80H
5BCB 01 DB 1
INIT4: ;INIT ROUTINE
5BCC C9 RET ;DONE

;
;CONFIGURATION 5-- IMSAI MIO SERIAL I/O
5BCD 4342420202CNFG5: DB 43H,42H,42H,2,2,1,1
5BD4 04 DB 4 ;INIT LENGTH
INIT5: ;INIT ROUTINE
5BD5 AF XRA A
5BD6 D343 OUT 43H
5BD8 C9 RET ;DONE

;
;CONFIGURATION 6 -- ALTAIR 88-4PIO
; OR OTHER PIO USING THE MOTOROLA 6820 PIA
5BD9 1011138080CNFG6: DB 16,17,19,80H,80H,3,3
5BE0 11 DB 17 ;INIT LENGTH
INIT6: ;INIT ROUTINE
5BE1 AF XRA A ;SELECT DATA DIRECTION
5BE2 D310 OUT 16 ;REGISTER AND SET
5BE4 D311 OUT 17 ;PORT A=IN
5BE6 D312 OUT 18 ;PORT B=OUT
5BE8 2F CMA
5BE9 D313 OUT 19 ;CA2/CB2=OUTPUT
5BEB 3E24 MVI A,24H ;LEVEL HANDSHAKE
5BED D310 OUT 16
5BEF D312 OUT 18
5BF1 C9 RET ;DONE

;
;
;*****
;*** SPECIAL CONFIGURATIONS ***
;*****
;
;
;*****
;SPECIAL CONFIGURATION 80 -- COMPAL 80
; TERMINAL I/O DONE THROUGH THE COMPAL MONITOR
; INTERRUPTS MUST BE ENABLED
;*****
;
0016 = MSIZE EQU 22 ;DECIMAL SIZE OF SYSTEM
5600 = BIOS EQU (MSIZE*1024)-200H ;STANDARD CP/M CONVENTION
5B00 = USER EQU BIOS+500H ;LOCATION TO USER AREA
C900 = OFFSET EQU 1F00H-BIOS ;TO SYSGEN IMAGE
;JUMPS MUST REMAIN HERE, IN SAME ORDER

```

CP/M on North Star Double Density Disk Users Notes

```

B00
;
; PAL0:                                ;COMPAL USER AREA JUMP TABLE
5B00 C31D5B      JMP      PALCINIT
5B03 C3295B      JMP      PALST
5B06 C31E5B      JMP      PALCI
5B09 C3245B      JMP      PALCO
5B0C C3245B      JMP      PALLIST
5B0F C3245B      JMP      PALPUNCH
5B12 C31E5B      JMP      PALREADER
5B15 C3325B      JMP      PALPRST
5B18 000000      DB       0,0,0
5B1B 3400        DW       PALLN          ;LENGTH
;
5B1D C9          RET                      ;INIT BY COMPAL MONITOR
;
PALCI:
PALREADER:
5B1E CDADE0      CALL     0E0ADH          ;CHR IN COMPAL MONITOR
5B21 E67F        ANI      7FH
5B23 C9          RET
;
PALCO:
PALLIST:
PALPUNCH:
5B24 79          MOV      A,C
5B25 CDC2E0      CALL     0E0C2H          ;CHR OUT COMPAL MONITOR
5B28 C9          RET
;
PALST:
5B29 FB          EI
5B2A 3AFDED      LDA      0EDFDH          ;CHR FROM INTERRUPT KEYBRD
5B2D B7C8        ORA      A ! RZ
5B2F 3EFF        MVI      A,0FFH
5B31 C9          RET
;
PALPRST:
5B32 AFC9        XRA      A ! RET
0034 =           PALLN    EQU      $-PAL0
;
;
;*****
;SPECIAL CONFIGURATION 81 -- PROCESSOR TECHNOLOGY SOL-20
;                               WITH SOLOS 1.3
;*****
;
0016 =           MSIZE    EQU      22          ;DECIMAL SIZE OF SYSTEM
5600 =           BIOS     EQU      (MSIZE*1024)-200H ;STANDARD CP/M CONVENTION
5B00 =           USER     EQU      BIOS+500H      ;LOCATION TO USER AREA
C900 =           OFFSET   EQU      1F00H-BIOS      ;TO SYSGEN IMAGE
;JUMPS MUST REMAIN HERE, IN SAME ORDER
5B00             ORG      USER
;
;
; SOL0:                                ;SOL USER AREA JUMP TABLE
5B00 C31E5B      JMP      SCINIT
5B03 C33B5B      JMP      SCONST
5B06 C3715B      JMP      SCONIN
5B09 C39C5B      JMP      SCONOUT
5B0C C3A25B      JMP      SLIST

```

CP/M on North Star Double Density Disk Users Notes

```

5B0F C3AA5B      JMP      SPUNCH
5B12 C3775B      JMP      SREADER
5B15 C3E05B      JMP      SPRSTAT
5B18 000000      DB       0,0,0
5B1B EE5B        DW       SOLLEN          ;LENGTH
;
;*** SOL EQUATES ***
C000 =          SOLOS   EQU      0C000H
C800 =          SOLRAM  EQU      SOLOS+800H
C019 =          SOUT    EQU      SOLOS+19H
C01C =          AOUT    EQU      SOUT+3
C01F =          SINP    EQU      AOUT+3
C022 =          AINP    EQU      SINP+3
C800 =          UIPRT   EQU      SOLOS+800H      ;STORE LOC OF USER INPUT PORT
C802 =          UOPRT   EQU      SOLOS+802H      ;STORE LOC OF USER OUTPUT PORT
;
;EQUATES FOR SELECTION OF SOL PSEUDO-PORTS
0000 =          CON     EQU      0              ;KEYBOARD AND VIDEO
0001 =          RDR     EQU      1              ;SERIAL PORT
0001 =          PUN     EQU      1              ;SERIAL PORT
0002 =          LST     EQU      2              ;PARALLEL PORT
IOSET:          ;INITIAL IOBYT MAPPING STORED HERE
                ;CHANGE TO SELECT OTHER DEVICES
5B1D 94          DB      LST SHL 6 + PUN SHL 4 + RDR SHL 2 + CON
;
SCINIT:         ;INSERT YOUR COLD INIT HERE IF NEEDED
5B1E 000000      NOP ! NOP ! NOP              ;USER SPACE
SOLINIT:        ;INIT FOR SOL-20
5B21 AF          XRA     A
5B22 D3FA        OUT     0FAH                  ;SHUT OFF SOL TAPES
5B24 3A1D5B      LDA     IOSET                  ;PICK UP INITIAL IOBYT
5B27 320300      STA     IOBYT                  ;STORE INITIAL IOBYT MAPPING
;
;BE SURE TO INITIALIZE SOLOS IF YOUR DEVICES ARE MAPPED
;INTO THE IOBYT - USE THIS ROUTINE WITH YOUR OWN VECTORS.
00C9 =          USERIN EQU      0C9H          ;RETURNS UNTIL
00C9 =          USEROUT EQU     0C9H          ;DEVICES SETUP
5B2A 21C9002200  LXI H,USERIN ! SHLD UIPRT
5B30 21C9002202  LXI H,USEROUT ! SHLD UOPRT
5B36 C9          RET                          ;END OF INITIALIZATION
;
SWINIT:         ;PUT YOUR WARM INIT ROUTINE HERE
5B37 000000      NOP ! NOP ! NOP              ;USER SPACE
5B3A C9          RET
;
SCONST:         ;PUT YOUR STATUS ROUTINE HERE
                ;RET Z SET AND A=0 IF NO KEY
                ;RET NZ AND A=0FFH IF KEY
5B3B 3A0300      LDA     IOBYT
5B3E E603        ANI     3
SOLST:          ;GENERAL SOL CONSOLE STATUS
                ;ENTER WITH ACC= 0-3 FOR DEVICE SELECT
                ;STORE CHAR INPUTTED (OR 80H IF NO KEY)
5B40 22EA5B      SHLD    SAVHL                  ;SAVE HL
5B43 60          MOV     H,B

```

CP/M on North Star Double Density Disk Users Notes

```

5B44 69      MOV      L,C
5B45 22EC5B  SHLD     SAVBC      ;AND BC REGS
5B48 4F      MOV      C,A      ;DEVICE SELECT
5B49 0600    MVI      B,0
5B4B 21E65B  LXI      H,CHAR0
5B4E 09      DAD      B      ;POINT TO CHAR FOR DEVICE
5B4F 22E45B  SHLD     PTCHAR
5B52 7E      MOV      A,M      ;80H IF NO CHAR
5B53 FE80    CPI      80H      ;CHAR THERE?
5B55 C2665B  JNZ      CSKEY      ;YES
                    ;READ THE PORT IF NO CHAR IN MEMORY
5B58 79      MOV      A,C      ;PSEUDO-PORT
5B59 CD22C0  CALL     AINP
5B5C 77      MOV      M,A      ;STORE CHAR
5B5D C2665B  JNZ      CSKEY      ;CHAR RCVD
5B60 3680    MVI      M,80H    ;NO CHAR FLAG
5B62 AF      XRA      A      ;A=0 IF NO KEY
5B63 C3685B  JMP      CSEXIT      ;NO KEY
5B66 3EFF    CSKEY: MVI      A,0FFH ;A=0FFH IF KEY
                    ;FLAGS ARE SET
5B68 2AEC5B  LHLD     SAVBC      ;RESTORE
5B6B 44      MOV      B,H      ;BC
5B6C 4D      MOV      C,L      ;AND
5B6D 2AEA5B  LHLD     SAVHL    ;HL REGS
5B70 C9      RET

;
;SCONIN:      ;CONSOLE INPUT DRIVER
5B71 3A0300  LDA      IOBYT
5B74 C37F5B  JMP      SOLIN

;
;SREADER:    ;READER INPUT
5B77 3A0300  LDA      IOBYT
5B7A 1F1F    RAR      ! RAR
5B7C C37F5B  JMP      SOLIN

;
;SOLIN:      ;GENERAL SOL INPUT
                    ;ACC HAS DEV SELECT
                    ;SAVE
5B7F E603    ANI      3
5B81 32E35B  STA      DEVSEL
5B84 CD405B  CALL     SOLST
5B87 3AE35B  LDA      DEVSEL
5B8A CA7F5B  JZ       SOLIN      ;UNTIL CHAR READY
5B8D 22EA5B  SHLD     SAVHL    ;SAVE HL REG
5B90 2AE45B  LHLD     PTCHAR
5B93 7E      MOV      A,M      ;GET CHAR
5B94 3680    MVI      M,80H    ;NO CHAR FLAG
5B96 2AEA5B  LHLD     SAVHL    ;RESTORE HL REG
5B99 E67F    ANI      7FH      ;STRIP PARITY
5B9B C9      RET

;
;SCONOUT:    ;CONSOLE OUTPUT DRIVER
5B9C 3A0300  LDA      IOBYT
5B9F C3B45B  JMP      SOLOUT

;
;SLIST:      ;LIST OUTPUT
5BA2 3A0300  LDA      IOBYT

```

CP/M on North Star Double Density Disk Users Notes

```

5BA5 0707          RLC ! RLC
5BA7 C3B45B        JMP      SOLOUT

;
SPUNCH:            ;PUNCH OUTPUT
5BAA 3A0300        LDA      IOBYT
5BAD 1F1F1F1F      RAR ! RAR ! RAR ! RAR
5BB1 C3B45B        JMP      SOLOUT

;
SOLOUT:            ;GENERAL SOL OUTPUT
                    ;ACC HAS DEVICE SELECT
                    ;STORE HL
5BB4 22EA5B        SHLD     SAVHL
5BB7 60            MOV      H,B
5BB8 69            MOV      L,C
5BB9 22EC5B        SHLD     SAVBC      ;AND BC
5BBC 41            MOV      B,C      ;SOLOS WANTS CHAR IN B
5BBD E603          ANI      3        ;DEV SELECT
5BBF C2D35B        JNZ      OUTSOL    ;NOT VIDEO
5BC2 3AE25B        LDA      LASTOUT   ;CHAR TO OUTPUT IN C
5BC5 FE0D          CPI      0DH      ;ASCII CARRIAGE RETURN
5BC7 C2CE5B        JNZ      OUTVID    ;CHECK IF 2 X CR
5BCA B9            CMP      C
                    ;DON'T OUTPUT TWO CARRIAGE RETURNS
5BCB CAD65B        JZ       OUTEXIT
OUTVID:            MOV      A,C
5BCE 79            STA      LASTOUT   ;SAVE IT
5BCF 32E25B        XRA      A        ;SET A=0 FOR VIDEO
5BD2 AF            OUTSOL:           ;SEND THE CHAR TO SOLOS
5BD3 CD1CC0        CALL     AOUT
OUTEXIT:           ;RESTORE REGS AND EXIT
                    ;PICK UP BC
5BD6 2AEC5B        LHLD     SAVBC
5BD9 44            MOV      B,H
5BDA 4D            MOV      C,L
5BDB 2AEA5B        LHLD     SAVHL     ;AND HL REGS
5BDE 79            MOV      A,C      ;SOME PROGS EXPECT CHAR
5BDF C9            RET              ;PUT OUT TO RET IN ACC
                    ;PRINTER STATUS
5BE0 AFC9          SPRSTAT: XRA A ! RET

;
;*** DATA AREA ***
5BE2 00            LASTOUT:DB 0      ;STORES POSS CARRIGE RET
5BE3              DEVSEL: DS 1      ;TO SELECT PSEUDO-PORT
5BE4              PTCHAR: DS 2      ;POINTER TO CHAR 0-3
5BE6 00            CHAR0:  DB 0      ;LASTIN FROM PSEUDO-PORT 0
5BE7 00            CHAR1:  DB 0      ;LASTIN FROM 1
5BE8 00            CHAR2:  DB 0      ;LASTIN FROM 2
5BE9 00            CHAR3:  DB 0      ;LASTIN FROM 3
5BEA              SAVHL:  DS 2      ;SAVE HL REG
5BEC              SAVBC:  DS 2      ;SAVE BC REG
5BEE =             SOLLEN  EQU $      ;LENGTH

;
;
;*****
;SPECIAL CONFIGURATION 82 -- TDL SMB MONITOR BOARD
;*****

```

```

;
0016 = MSIZE EQU 22 ;DECIMAL SIZE OF SYSTEM
5600 = BIOS EQU (MSIZE*1024)-200H ;STANDARD CP/M CONVENTION
5B00 = USER EQU BIOS+500H ;LOCATION TO USER AREA
C900 = OFFSET EQU 1F00H-BIOS ;TO SYSGEN IMAGE
;JUMPS MUST REMAIN HERE, IN SAME ORDER
5B00 ORG USER
SMB0: ;SMB USER AREA
5B00 C31D5B JMP SMBCINIT
5B03 C3335B JMP SMBST
5B06 C31E5B JMP SMBCI
5B09 C32A5B JMP SMBCO
5B0C C3305B JMP SMBLO
5B0F C32D5B JMP SMBPO
5B12 C3245B JMP SMBRI
5B15 C3365B JMP SMBPRST
5B18 000000 DB 0,0,0
5B1B 3800 DW SMLEN
SMBCINIT: ;INIT DONE BY SMB
5B1D C9 RET
SMBCI:
5B1E CD03F0 CALL 0F003H ;CI ENTRY IN SMB
5B21 E67F ANI 7FH
5B23 C9 RET
5B24 CD06F0 SMBRI: CALL 0F006H ;RI ENTRY IN SMB
5B27 E67F ANI 7FH
5B29 C9 RET
5B2A C309F0 SMBCO: JMP 0F009H ;CO ENTRY IN SMB
5B2D C30CF0 SMBPO: JMP 0F00CH ;PO ENTRY IN SMB
5B30 C30FF0 SMBLO: JMP 0F00FH ;LO ENTRY IN SMB
5B33 C312F0 SMBST: JMP 0F012H ;CSTS ENTRY IN SMB
SMBPRST:
5B36 AFC9 XRA A ! RET
0038 = SMLEN EQU $-SMB0
;
;
;*****
;SPECIAL CONFIGURATION 83 - NORTH STAR HORIZON
;*****
;
0016 = MSIZE EQU 22 ;DECIMAL SIZE OF SYSTEM
5600 = BIOS EQU (MSIZE*1024)-200H ;STANDARD CP/M CONVENTION
5B00 = USER EQU BIOS+500H ;LOCATION TO USER AREA
C900 = OFFSET EQU 1F00H-BIOS ;TO SYSGEN IMAGE
;JUMPS MUST REMAIN HERE, IN SAME ORDER
5B00 ORG USER
HOR0: ;BASE OF HORIZON USER AREA
5B00 C3A65B JMP HORCINIT
5B03 C31D5B JMP HORST
5B06 C33B5B JMP HORIN
5B09 C3735B JMP HOROUT
5B0C C3735B JMP HORLIST
5B0F C3735B JMP HORMPUNCH
5B12 C33B5B JMP HORREADER
5B15 000000 DB 0,0,0

```

CP/M on North Star Double Density Disk Users Notes

```

5B18 FC00          DW          HORLEN
5B1A 00          HORIOB: DB          0          ;IOBYTE FOR HORIZON
5B1B 00          PARFLG: DB          0          ;0=NO PAR, 1=PARITY
                                           ;SET ON MEMORY BOARD
5B1C E8          PROMLOC:DB          0E800H/256 AND 0FFH      ;ADDR OF PROM
HORST:
5B1D 3A1A5B      LDA          HORIOB
5B20 E603          ANI          3          ;STRIP OUT CONSOLE
5B22 CA2B5B      JZ          HORST0
5B25 3D          DCR          A
5B26 CA335B      JZ          HORST1
5B29 AFC9          XRA A ! RET          ;ONLY 2 SERIAL PORTS
5B2B DB03          HORST0: IN          3
5B2D E602          ANI          2
5B2F C8          RZ          ;NO KEY
5B30 3EFFC9      MVI A,0FFH ! RET      ;KEY
5B33 DB05          HORST1: IN          5
5B35 E602          ANI          2
5B37 C8          RZ          ;NO KEY
5B38 3EFFC9      MVI A,0FFH ! RET      ;KEY
HORREADER:
HORIN:          ;HORIZON INPUT ROUTINE
5B3B 3A1A5B      LDA          HORIOB
5B3E E603          ANI          3
5B40 CA5B5B      JZ          HORIN0      ;1ST SERIAL PORT
5B43 3D          DCR          A
5B44 CA675B      JZ          HORIN1      ;2ND SERIAL PORT
5B47 3D          DCR          A
5B48 C0          RNZ          ;NO 4TH DEVICE
                                           ;FALL INTO PARALLEL ROUTINE
HORIN2:          ;PARALLEL PORT
5B49 DB06          IN          6          ;MOTHERBOARD STATUS
5B4B E602          ANI          2          ;MASK TO GET PI FLAG
5B4D CA495B      JZ          HORIN2
5B50 DB00          IN          0          ;READ KEYBOARD
5B52 F5          PUSH         PSW        ;SAVE CHAR
5B53 3E30          MVI          A,30H
5B55 D306          OUT          6          ;RESET PI FLAG
5B57 F1E67F      POP PSW ! ANI 7FH
5B5A C9          RET
5B5B DB03          HORIN0: IN          3          ;FIRST SERIAL PORT
5B5D E602          ANI          2
5B5F CA5B5B      JZ          HORIN0
5B62 DB02          IN          2
5B64 E67F          ANI          7FH
5B66 C9          RET
5B67 DB05          HORIN1: IN          5          ;SECOND SERIAL PORT
5B69 E602          ANI          2
5B6B CA675B      JZ          HORIN1
5B6E DB04          IN          4
5B70 E67F          ANI          7FH
5B72 C9          RET
;
HORLIST:
HORPUNCH:

```

```

5B73 3A1A5B      LDA      HORIOB
5B76 E603         ANI      3
5B78 CA905B      JZ       HOROUT0      ;1ST SERIAL PORT
5B7B 3D          DCR      A
5B7C CA9B5B      JZ       HOROUT1      ;2ND SERIAL PORT
5B7F 3D          DCR      A
5B80 C0          RNZ
                                ;ONLY 3 PORTS
                                ;FALL INTO PARALLEL OUTPUT ROUTINE
                                ;PARALLEL OUTPUT
                                ;MOTHERBOARD STATUS
                                ;MASK TO GET PO FLAG

5B81 DB06        IN       6
5B83 E601        ANI      1
5B85 CA815B      JZ       HOROUT2
5B88 79          MOV      A,C
5B89 D300        OUT      0      ;OUTPUT CHAR
5B8B 3E20        MVI      A,20H
5B8D D306        OUT      6      ;RESET PO FLAG
5B8F C9          RET

5B90 DB03        IN       3
5B92 E601        ANI      1
5B94 CA9B5B      JZ       HOROUT1
5B97 79          MOV      A,C
5B98 D302        OUT      2
5B9A C9          RET

5B9B DB05        IN       5
5B9D E601        ANI      1
5B9F CA9B5B      JZ       HOROUT1
5BA2 79          MOV      A,C
5BA3 D304        OUT      4
5BA5 C9          RET

;
HORCINIT:
5BA6 3EFE        MVI      A,0FEH
5BA8 D3C0        OUT      0C0H      ;DISARM PARITY
5BAA 3A1B5B      LDA      PARFLG
5BAD B7          ORA      A
5BAE CADA5B      JZ       NOPAR
                                ;
                                ;SPECIAL INIT ROUTINE FOR HORIZON
                                ;COME HERE IF YOU USE THE PARITY OPTION
                                ;
5BB1 210000      TINIT:   LXI      H,0
5BB4 3A1C5B      LDA      PROMLOC
5BB7 57          MOV      D,A
5BB8 7C          TINKL:   MOV      A,H
5BB9 BA          CMP      D
5BBA C2C35B      JNZ      TINCP
5BBD C604        ADI      4      ;SKIP 400H OVER PROM
5BBF 67          MOV      H,A
5BC0 CAD65B      JZ       TINU
5BC3 7E          TINCP:   MOV      A,M
5BC4 77          MOV      M,A
5BC5 2C          INR      L

```



```

5B12 C3255B      JMP      VECREADER
5B15 C3475B      JMP      VECPRST
5B18 000000      DB       0,0,0
5B1B 4A00        DW       VECLEN

VECST:           ;CONSOLE STATUS
                 ;LOOK FOR CHAR

5B1D CD315B      CALL     VECSTAT
5B20 3E00        MVI      A,0
5B22 C8          RZ       ;NO KEY, EXIT
5B23 2F          CMA      ;IF KEY
5B24 C9          RET      ;EXIT

VECREADER:
VECIN:           ;CONSOLE INPUT
                 ;SEE IF CHAR READY
                 ;LOOP UNTIL CHAR
                 ;CHAR STORED BY VECSTAT
                 ;TELL STAT RTN WE HAVE CHAR
                 ;STRIP PARITY
                 ;EXIT

5B25 CD315B      CALL     VECSTAT
5B28 CA255B      JZ       VECIN
5B2B 7E          MOV      A,M
5B2C 3600        MVI      M,0
5B2E E67F        ANI      7FH
5B30 C9          RET

VECSTAT:         ;POINT TO STORAGE
                 ;THIS IS 0 IF
                 ;NO CHAR THERE PREV
                 ;RET IF CHAR ALREADY THERE
                 ;PROM INPUT RTN
                 ;STORE CHAR
                 ;FLAG

5B31 21495B      LXI      H,VCHAR
5B34 34          INR      M
5B35 35          DCR      M
5B36 C0          RNZ
5B37 CDDCC0      CALL     CNTLC
5B3A 77          MOV      M,A
5B3B B7          ORA      A
5B3C C9          RET

VECLIST:
VECPUNCH:
VECOUT:          ;OUTPUT ROUTINE
                 ;PROM WANTS CHAR IN ACC
                 ;PROM OUTPUT RTN
                 ;EXIT
                 ;COLD INIT

5B3D 79          MOV      A,C
5B3E CD98C0      CALL     PTLOP
5B41 C9          RET

VECCINIT:        ;WARM INIT

5B42 AF          XRA      A
5B43 32495B      STA      VCHAR
5B46 C9          RET

VECPRST:         ;PRINTER STATUS
                 ;STORE CHAR FROM VECSTAT

5B47 AFC9        XRA A ! RET
5B49 00          DB       0
004A =           EQU      $-VECO
;
;

```

DIGITAL RESEARCH
Box 579 Pacific Grove, California, 93950
SOFTWARE LICENSE AGREEMENT

IMPORTANT: All Digital Research programs are sold only on the condition that the purchaser agrees to the following license. **READ THIS LICENSE CAREFULLY.** If you do not agree to the terms contained in this license, return the packaged diskette **UNOPENED** to your distributor and your purchase price will be refunded. If you agree to the terms contained in this license, fill out the **REGISTRATION** information and **RETURN** by mail.

DIGITAL RESEARCH agrees to grant and the Customer agrees to accept on the following terms and conditions nontransferable and nonexclusive licenses to use the software program(s) (Licensed Programs) herein delivered with this agreement.

TERM:

This agreement is effective from the date of receipt of the above-referenced program(s) and shall remain in force until terminated by the Customer upon one month's prior written notice, or by Digital Research as provided below.

Any license under this Agreement may be discontinued by the Customer at any time upon one month's prior written notice. Digital Research may discontinue any license or terminate this Agreement if the Customer fails to comply with any of the terms and conditions of this Agreement.

LICENSE:

Each program license granted under this Agreement authorizes the Customer to use the Licensed Program in any machine readable form on any single computer system (referred to as System). A separate license is required for each System on which the Licensed Program will be used.

This Agreement and any of the licenses, programs or materials to which it applies may not be assigned, sublicensed or otherwise transferred by the Customer without prior written consent from Digital Research. No right to print or copy, in whole or in part, the Licensed Programs is granted except as hereinafter expressly provided.
PERMISSION TO COPY OR MODIFY LICENSED PROGRAMS:

The customer shall not copy, in whole or in part, any Licensed Programs which are provided by Digital Research in printed form under this Agreement. Additional copies of printed materials may be acquired from Digital Research.

Any Licensed Programs which are provided by Digital Research in machine readable form may be copied, in whole or in part, in printed or machine readable form in sufficient number for use by the Customer with the designated System, to understand the contents of such machine readable material, to modify the Licensed Program as provided below, for back-up purposes, or for archive purposes, provided, however, that no more than five (5) printed copies will be in existence under any license at any one time without prior written consent from Digital Research. The Customer agrees to maintain appropriate records of the number and location of all such copies of Licensed Programs. The original, and any copies of the Licensed Programs, in whole or in part, which are made by the Customer shall be the property of Digital Research. This does not imply, of course, that Digital Research owns the media on which the Licensed Programs are recorded. The Customer may modify any machine readable form of the Licensed Programs for his own use and merge it into other program material to form an updated work, provided that, upon discontinuance of the license for such Licensed Program, the Licensed Program supplied by Digital Research will be completely removed from the updated work. Any portion of the Licensed Program included in an updated work shall be used only if on the designated System and shall remain subject to all other terms of this Agreement.

The Customer agrees to reproduce and include the copyright notice of Digital Research on all copies, in whole or in part, in any form, including partial copies of modifications, of Licensed Programs made hereunder.

PROTECTION AND SECURITY:

The customer agrees not to provide or otherwise make available any Licensed Program including but not limited to program listings, object code and source code, in any form, to any person other than Customer or Digital Research employees, without prior written consent from Digital Research, except with the Customer's permission for purposes specifically related to the Customer's use of the Licensed Program.

DISCONTINUANCE:

Within one month after the date of discontinuance of any license under this Agreement, the Customer will furnish Digital Research a certificate certifying the through his best effort, and to the best of his knowledge, the original and all copies, in whole or in part, in any form, including partial copies in modifications, of the Licensed Program received from Digital Research or made in connection with such license have been destroyed, except that, upon prior written authorization from Digital Research, the Customer may retain a copy for archive purposes.

DISCLAIMER OF WARRANTY:

Digital Research makes no warranties with respect to the Licensed Programs. The sole obligation of Digital Research shall be to make available all published modifications or updates made by Digital Research to Licensed Programs which are published within one (1) year from date of purchase, provided Customer has returned the Registration Card delivered with the Licensed Program.

LIMITATION OF LIABILITY:

THE FOREGOING WARRANTY IS IN LIEU OF ALL OTHER WARRANTIES, EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. IN NO EVENT WILL DIGITAL RESEARCH BE LIABLE FOR CONSEQUENTIAL DAMAGES EVEN IF DIGITAL RESEARCH HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

GENERAL

If any of the provisions, or portions thereof, of this Agreement are invalid under any applicable statute or rule of law, they are to that extent to be deemed omitted.

LIFEBOAT ASSOCIATES
2248 Broadway Suite 34
NEW YORK, NY 10024
(212) 580-0082



MICROPRO INTERNATIONAL CORPORATION

END - USER AGREEMENT

WHEREAS, MicroPro International Corporation of 5810 Commerce Boulevard, Rohnert Park, California, hereinafter referred to as Company, develops proprietary computer programs and sells use licenses for such proprietary computer programs together with or apart from accompanying copyrighted media material and documentation and;

5810 Commerce Boulevard
Rohnert Park, CA 94928

(707) 544-2865

Local Area Telephones:

(415) 398-7062
(916) 485-7619
(213) 224-1619
(714) 634-2908
(408) 279-8980
(209) 445-0511

WHEREAS, _____ of _____
City of _____ County of _____

State of _____ Country of _____, hereinafter referred to as End User, desires to obtain the benefits and, in return for which, is willing to abide by the obligations and fee agreements of Company's use licenses in Company's proprietary computer programs;

NOW THEN, for good and valuable consideration, including but not limited to license grant by Company to End User and End User's covenant regarding Company's proprietary rights, Company agrees to sell End User useable materials representing Company's product or products subject to the following terms and conditions:

W I T N E S S E T H

ARTICLE I. EXCLUSIVE SOURCE. End User shall obtain all Company product materials through Company or Company Authorized Dealer and no other source. Company product materials include, but are not limited to, manuals, license agreements and media upon which Company's proprietary computer programs are recorded. Except for archival copies, as defined in ARTICLE III of this Agreement, End User shall make no copies, of any kind, of any of the materials furnished by Company, unless specifically authorized to do so in writing signed by an officer of Company.

ARTICLE II. LIQUIDATED DAMAGES. End User recognizes that Company has gone through considerable time and expense to develop Company's products and Company would be damaged by unauthorized copying or reproduction or distribution of Company's product materials. In the event End User breaches this Agreement by unauthorized copying or reproducing or distributing of Company's product materials, End User agrees to pay Company as liquidated damages, for each occurrence of the unauthorized act of copying or reproducing or distributing of Company product materials, the sum of \$1,500.00 (One thousand five hundred dollars) in lawful United States Currency, immediately upon demand by Company.

Furthermore, Company agrees to pay to any person, company, or other lawfully constituted entity that provides information that leads to a successful prosecution and recovery, the sum of \$1,000 or half the amount thus recovered, whichever is greater. At all times, the decision to evaluate such information as well as the decision to prosecute remains at the sole option of Company. Furthermore, any sum paid will only be paid once, and, in the event of multiple entities, will be distributed at the sole discretion of Company.

ARTICLE III. ARCHIVAL COPIES. End User may make archival copies of those portions of Company's product(s) that are provided on machine readable media, provided such copies are for the End User's personal use and that no more than one such copy is in use at any time unless End User has paid for multiple copy use as described in ARTICLE IV of this Agreement.

ARTICLE IV. MULTIPLE COPY USE. Company use licenses are applicable to a single microcomputer installation. In the event End User intends to use Company Product or any part thereof on more than one microcomputer, the required fee for such multiple use must be paid.

ARTICLE V. LIMITED WARRANTY POLICY. Company warrants that all materials furnished by Company constitutes an accurate manufacture of Company product and will replace any such Company furnished material found to be thus defective, provided such defect is found within ten days of purchase by End User. However, Company makes NO express or implied warranty of any kind with regard to performance or fitness of purpose for any Company product. Furthermore, Company is NOT responsible for any loss or inaccuracy of data of any kind nor for any consequential damages resulting therefrom whether through Company negligence or not. Company will not honor any warranty where Company Product has been subjected to physical abuse or used in defective or non-compatible equipment.

ARTICLE VI. UPDATE POLICY. Company may, from time to time, revise the performance of its products and, in so doing, incur NO obligation to furnish such revisions to any Company customer. At Company's option, Company may provide such revisions to its Dealers. Also, at Company's option, Company may provide its Dealers and End Users with a revision newsletter from time to time.

ARTICLE VII. GOVERNING LAW. This Agreement shall be interpreted in accordance with the laws of the State of California. In the event any part of this Agreement is invalidated by court or legislative action, the remainder of this Agreement shall remain in blinding effect.

ARTICLE VIII. LEGAL FEES. In the event of legal action brought by either party, the prevailing party shall be entitled to reimbursement of legal fees as set by court action.

ARTICLE IX. ENTIRE AGREEMENT. This Agreement constitutes the entire agreement between the parties and supersedes any prior agreements. This Agreement may only be changed by mutual written consent.

Date _____

for Company or Company Authorized Dealer

Company Product Name:
Valid End User Serial Nos.:

Company Product Name:
Valid End User Serial Nos.:

Company Product Name:
Valid End User Serial Nos.:

Company Product Name:
Valid End User Serial Nos.:

Orders for the products of Microsoft and MicroPro must be accompanied by executed license agreements. If you care to return these immediately, Lifeboat Associates will hold them on file and permit us to accept an order without delay at a subsequent date. Return of these forms does not obligate you to order any materials.

Return these forms to:

Lifeboat Associates
2248 Broadway
New York, N.Y. 10024

Microsoft Software Non-Disclosure Agreement

The party below agrees that it is receiving Microsoft Software for use on a single computer only, as designated on the registration card(s). The party agrees to fill out and mail in the registration card(s) before making use of the Microsoft Software. The party agrees that all copies of Microsoft Software are owned by Microsoft, that all copies will display Microsoft's copyright, that all copies will be strictly safeguarded against disclosure to or use by persons not authorized by Microsoft to use such Microsoft Software, and that the location of all copies will be reported to Microsoft at Microsoft's request. The party may make up to two additional copies, for back-up purposes, of each Software Product received under this agreement. The Party agrees that unauthorized copying or disclosure will cause great damage to Microsoft. The Party agrees that the terms of this Agreement will extend to each Microsoft Software Product received from Lifeboat Associates.

Signature

Print Name

Print Company Name

Print Title

Print Address and Phone Number

CP/M on North Star Disk - Double Density Version

Errata in Users Notes Rev 1.0 dated 2/24/79

- 1) In section VI, paragraph 3, the third and fourth sentences should read:

Just give the command "FORMAT CR" and the program will ask you which drive to use and whether to format in single or double density. The disk is ready to use after FORMAT runs.

- 2) The section on page 12 titled 'Using Existing Single Density User Area' is in error and should not be followed. Any owner of a single density system should use their editor and assembler to create a suitable USER area. This may be the same as used in the single density system, but must be specially ORGed at 5B00H.

If this is prepared, the procedure to place it in the second half of sector 17 on the Double Density disk may be followed.

- 3) In the last paragraph on page 17, the command:

SAVE 36 CPM24.COM

should be:

SAVE 36 CPM22.COM

- 4) Two data lines in the listing Appendix are incorrect. In configuration number 3 the line:

0,0,0,1,1,20H,20H,20H,20H,2,2

should be:

0,1,1,20H,20H,2,2 for TTS,TTI,TTO,RDA,RDAMSK,TBE,TBEMSK

In configuration number 4, the line:

0,0,0,1,1,40H,40H,40H,40H,80H,80H

should be:

0,1,1,40H,40H,80H,80H for TTS,TTI,TTO,RDA,RDAMSK,TBE,TBEMSK

- 5) The Horizon configuration has been slightly altered. The disk file GENUSER.ASM has the correct code as used in the system for configurations 3 and 4 as described above and the revised Horizon routines.

March 8, 1979

Additional Notes on Double Density
CP/M on North Star

Based on enquiries, the following additional notes are offered to users of Double Density CP/M on North Star:

- 1) A diskette must be formatted using the FORMAT program before it can be used with CP/M. The one exception is that the COPY program will format a new or damaged disk. If you wish to take advantage of this feature, you must copy ALL the diskette tracks to prevent some remaining unformatted.
- 2) Certain software products of Lifeboat Associates may be shipped to double density users on single density format. If the label does not explicitly state double density format, use the DENSITY program to set drive B to single density and use PIP to copy the files to the double density diskette on drive A.
- 3) After running the MOVCPM program and SYSGENing the enlarged system onto a disk, the new system is unconfigured. It is necessary to cold boot such a system the first time with the program CONFIG.COM on the diskette. Patching the byte at 120H to the correct configuration number and executing at 100H will cause the system to sign-on and then SAVEUSER may fix the patched USER area onto the diskette. The DOS method suggested for initial system start-up is only intended to be used on the 22K system as distributed.
- 4) The CONFIG.COM program can be permanently configured by patching the byte at 120H to the configuration number and re-saving the file on the disk. If this is done, cold booting a newly created system will cause CONFIG to immediately execute the correct sequence.
- 5) In the event that a highly non-standard USER section is required and no monitor is available, the DOS method will work for the first time around. For other system sizes the USER, correctly assembled for the new size, may be inserted in CONFIG at 200H. If the entry of CONFIG at 100H is patched to a jump to 109H, CONFIG will move the USER routines to the correct location in BIOS, ready for the SAVEUSER execution.
- 6) A summary of CONFIG.COM entry points:
 - 100H Jump to 103H if the value at 120H is 0FFH. If 120H has a configuration number, then move the appropriate USER routines to the USER address in BIOS and sign-on.
 - 103H Jump to 103H. This is a "soft" halt and may be modified to re-enter a monitor, such as DDT at 38H or Zapple at 0F000H.
 - 106H Move the configured USER appropriate to the value set at 120H to memory starting at 200H and then jump to 103H.
 - 109H Move the routines in memory at 200H to the USER address in BIOS and then jump to sign-on.

Additional Notes on Double Density
CP/M on North Star

Based on a number of enquiries from users of the Double Density CP/M system, these notes are intended to present two procedures in a fuller dialog format. The first is a modification of the system initiation procedure suggested in the Users Notes which employs the feature that CONFIG will not soft halt but run to conclusion when the value byte has been pre-set to a valid configuration number. This is offered so that the system relocation procedure (PROCEDURE II) may be treated in a similar manner using the pre-setting of CONFIG from the first earlier session.

The second procedure is used to relocate the CP/M system to make use of a different (usually larger) memory size than the distribution system utilization of 24K of R/W memory.

PROCEDURE I. Bringing up CP/M by patching the CONFIG.COM program

Step 1. Boot in NORTHSTAR DOS in the usual way.

2. Insert copy (NEVER ORIGINAL) of CP/M diskette into drive A.

3. Type command: GO CPM (cr).

You should get the normal sign-on message:

CP/M on North Star double density disk

22K Version 1.44

Copyright (C) 1979 Lifeboat Associates

A>

4. Type command: DDT CONFIG.COM (cr). The system will respond with:

DDT VERS 1.4

NEXT PC

0B00 0100

5. Type: S120 (cr). The system will respond with:

0120 FF

6. Type the hex byte for your system (e.g. ah 83 for a HORIZON) followed by a (cr). The system will respond with a

0121 A0

7. Type: . (period) followed by a (cr). At this point we could simply type G100 to execute the configurator. Instead we will save the patched configurator so that it can also be used in Procedure II below for system relocation.

8. Type: ^C (control C). The system will return to the A> prompt.

9. Type: SAVE 10 CONFIG.COM (cr). Note that the 10 was from the 0B00 message of DDT. It refers to the 10 decimal pages occupied by CONFIG.COM between 100H and B00H in memory. The system configurator program, CONFIG.COM, has now been modified to preset to the desired configuration.

10. Now COLD boot the system. You should get the message:

CONFIG Version 1.5

Copyright (C) 1979 Lifeboat Associates

Your CP/M system is now configured.

Type SAVEUSER to permanently save on disk.

A>

11. Type: SAVEUSER (cr). The program will prompt you to finish the procedure by typing a (cr) to patch the system tracks on disk.

PROCEDURE II. System relocation.

As supplied, the CP/M system uses 24K of R/W memory starting at address 0. This is not sufficient to use the majority of the languages and applications software available. Attempts to run programs too large for the memory will result in a LOAD ERROR message. To utilize more memory it is necessary to employ the MOVCPM system relocation utility.

- Step 1. Copy the configured diskette from Procedure I. (NOTE: This procedure depends upon having a modified version of CONFIG.COM which is prepared as in steps 4 through 9 of Procedure I).
2. Place the copy in drive A and cold boot.
3. Type command: MOVCPM xx (cr) (xx is number of K two less than the amount of memory you wish to use. Type MOVCPM * to use all available memory). The system will report the system size being generated and end by inviting the use of SYSGEN or the saving of the system image as a file.
4. Type command: SYSGEN (cr). The SYSGEN program will sign-on:
SYSGEN VER 1.1
FOR NORTH STAR DD CP/M

SOURCE DRIVE (OR RETURN TO SKIP)

Type a (cr) since the system is in memory and not to be fetched from the system tracks of a diskette. The program will continue:

DESTINATION DRIVE NAME (OR RETURN TO REBOOT)

Type: A signifying drive A is to receive the system. SYSGEN will continue with the prompt:

DESTINATION ON A, THEN TYPE RETURN

The desired diskette is already mounted in drive A so simply hit the (cr). SYSGEN will offer another opportunity to write the system with:

FUNCTION COMPLETE

DESINATION DRIVE NAME (OR RETURN TO REBOOT)

At this point do NOT type a return, since a warm boot which would follow would fail, as the system in residence (22K) is now different from the system on the diskette. Perform a COLD boot. The system will announce the message:

CONFIG Version 1.5

Copyright (C) 1979 Lifeboat Associates

Your CP/M System is now configured.

Type SAVEUSER to permanently save on disk.

A>

5. Type command: SAVEUSER. The program will prompt you to type a (cr) to patch the configuration onto the disk system tracks.

A trial cold boot now will cause a system to sign on with the the correct new system size in the message.