820-II PERSONAL COMPUTER

CP/M 2.2 OPERATING SYSTEM REFERENCE GUIDE

HISTORY PAGE

This package contains supplement pages to be inserted in your 820-II CP/M 2.2 Personal Computer Reference Guide. To update your manual, insert this page as the first page in the manual, then remove and add new pages as instructed below.

MANUAL REORDER # 9R80448

		Part #	Software Level
Manual Issue Date:	5/1/82	79S80096A	1.000
Supplement Dates:	6/1/82	156P82598 (79S80096B)	1.000

This supplement updates the CP/M 2.2 documentation to the DCT000008 software level and includes the Rigid Disk documentation with the following page changes:

SECTION	REMOVE	ADD
Table of Contents	v thru viii	v thru viii
Introduction	1 through 8	1 through 8
General Programs	1 through 54	1 through 54
Systems Components	i,ii 1 thru 4 18,19 23 thru 32 35,36	i,ii 1 thru 4 18,19 23 thru 32 35,36 51,52
Reference	1,2 9 thru 16 21,22,31,32	1,2 9 thru 16 21,22,31,32
Diagnostics	1 thru 10	1 thru 10

CP/M 2.2 OPERATING SYSTEM REFERENCE GUIDE

820-II PERSONAL COMPUTER

9R80448

This equipment has been certified to comply with the limits for a Class B computing device, pursuant to Subpart J of part 15 of FCC Rules. Only peripherals (computer input/output devices, terminals, printers, etc.) certified to comply with the Class B limits may be attached to this computer. Operation with non-certified peripherals is likely to result in interference to radio and TV reception.

This equipment generates and uses radio frequency energy and if not installed and used properly, that is, in strict accordance with the manufacturer's instructions, may cause interference to radio and television reception. It has been type tested and found to comply with the limits for a Class B computing device in accordance with the specifications in Subpart J of part 15 of FCC Rules, which are designed to provide reasonable protection against such interference in a residential installation. However, there is no guarantee that interference will not occur in a particular installation. If this equipment does cause interference to radio or television reception, which can be determined by turning the equipment off and on, the user is encouraged to try to correct the interference by one or more of the following measures:

Reorient the receiving antenna.
Relocate the computer with respect to the receiver.
Move the computer away from the receiver.
Plug the computer into a different outlet so that computer and receiver are on different branch circuits.

If necessary, the user should consult the dealer or an experienced radio/television technician for additional suggestions. The user may find the following booklet prepared by the Federal Communications Commission helpful.

"HOW TO IDENTIFY AND RESOLVE RADIO-TV INTERFERENCE PROBLEMS"

This booklet is available from the U.S. GOVERNMENT PRINTING OFFICE, WASHINGTON, D.C. 20402, STOCK NO. 004-000-00345-4.

TABLE OF CONTENTS

GENERAL PROGRAMS

SYSTEM COMPONENTS

INTRODUCTION

This is your CP/M Operating System Reference Manual. An operating system is a set of programs that controls the computer's internal operation. In this manual you will find detailed instructions for using the CP/M Operating System on your XEROX 820-II Personal Computer.

The CP/M Operating System is an industry standard that lets you use a variety of programs you can purchase at software houses everywhere. Instructions for using application programs can be found in the CP/M Handbook.

If you have not used CP/M and the 820-II before, it is recommended that you go through the CP/M Handbook to learn how to operate the 820-II. This introduction section also gives you basic information about using the 820-II. You may want to read through it after you finish the handbook.

If you need more detailed information about the 820-II and CP/M, you'll find it in the other sections of this reference guide. A technical description of the 820-II and ROM monitor commands is given in the REFERENCE section of this manual. This information will be helpful to you as a programming aid.

One final note on using the 820-II successfully:

> **It is recommended that you <u>always</u> remove your disks from the 820-II before you turn it off. Leaving disks in the system when you power down can permanently erase information on the disks.**

DISK OPTIONS

The 820-II can utilize either single or double sided disk drives. If you're not sure what type of drive your system has, you can check the serial number plate (located on the bottom of the disk drives) and compare the number to the chart below. Page 3 shows the type of disk to use in each drive.

CAUTION: When checking the serial number on the Rigid Disk Drives, **don't** bump or drop the unit or you may damage the Rigid Disk.

Disk Drives	Serial Number	Disk Drives	Serial Number
5¼" Single Sided =	X929-000-0000	8" Single Sided =	X973-000-0000
5¼" Double Sided =	T66-000-0000	8" Double Sided =	F10-000-0000
		8" Rigid =	U07-000-0000

In addition to the number of sides on a disk, the 820-II will allow you to select the density of the information recorded on the disk. "Density" refers to how much data can be stored on a disk. The 820-II is designed to record in "double density", which means that you'll get twice the amount of data on the disk as "single density". You do have the option of selecting and using single density on the 820-II, if you so desire.

The density of a disk is determined when the disk is initialized. The INIT utility (step-by-step instructions for INIT are in the handbook) lets you select:

1 Single Density, Single Side
2 Single Density, Double Side
3 Double Density, Single Side
4 Double Density, Double Side

Usually, you'd select the density and number of sides you want to work with and initialize all your disks to work that way. This is the most convenient way to work and is recommended for anyone just learning the 820-II.

Read the rest of this page **only** when using:

- disks that have different densities.
- disks that have different numbers of sides.

Should this situation arise, use the following guidelines:

- Use the PIP program to copy files between disks with different densities or numbers of sides.
- Remember that the 820-II "sets" a disk drive to work in a certain density and number of sides when it reads the first disk you insert after loading the CP/M software. If you want to insert another disk with a different density and number of sides, you'll have to tell the 820-II to "reset" the drive in one of two ways:
  - You can press CTRL + C to reload the CP/M software, or
  - When using a floppy disk system you can tell the 820-II to pretend that the drive has a different name. The A Drive can be referred to as Drive **C** and the B Drive can be referred to as Drive D.

    For example: This means that you can put a double density disk in Drive B and work in double density by using **B** as the drive name (e.g., use B:filename to address a file on the disk). Then, when you remove that disk and insert a single density disk, you can refer to the drive as **D** (e.g., D:filename) so the 820-II will recognize the change in density. Using this method, you can switch back and forth between densities without reloading CP/M by referring to the drive as B and D.

Should you ever want to check to see what kind of disk a drive is set to read, you can use the WHATSA program. This program will display a list of the possible drive names (A through H) and the type of disk that is read (logged) by each drive. Remember, the first disk inserted and read by a drive after CP/M is loaded determines what type of disk the drive is set for.

It should be noted that the first drive logged on can be referred to as A or C. Another floppy disk drive can be referred to as B or D. The other drive names (E through H) are used in conjunction with a rigid disk.

FLOPPY DISKS

The disks used in the 820-II can be purchased from Xerox or from any computer or office equipment dealer. Disks will vary in quality and type. When a disk is marked "certified" for double density, it means that the surface is of a high enough quality to allow you to record double density data on the disk. Disks certified double sided are a high enough quality to prevent errors in recording on either side of the disk.

When purchasing 8" disks, specify:

- Either Single Sided (77 tracks) or **certified** Double Sided (77 tracks per side).
- Double Density **certified.**
- Soft Sectored.

When purchasing 5¼" disks, specify:

- Either Single Sided (40 tracks) or **certified** Double Sided (40 tracks per side).
- Double Density **certified.**
- Soft Sectored.

USING THE COPY UTILITY

The COPY utility will copy single or double density disks. The utility will physically copy disks that have the same density and the same number of sides (such as, single sided double density). When copying disks or files that have different densities or sides, you would use the PIP utility to copy your files from one disk to the other and the SYSGEN utility to copy the operating system.

If you suspect a problem with the files being copied, use the PIP utility with the verify option instead of using the copy utility (the copy utility will copy the problem along with the data).

CONFIGURING THE 820-II

You can make changes to certain areas of the CP/M software to customize the 820-II to your specific needs.

You can use the CONFIGUR or SET utility to change the software to operate with different applications. The following is a list of modifications that can be made to the software using the CONFIGUR utility:

- CP/M commands can be entered as restart commands to be executed each time that CP/M is loaded.
- The standard printer port options may be changed to operate with different printers other than the Xerox 620 (20 CPS) and 630 (40 CPS) printers.
- The communication port options may be changed.
- The I/O device assignments (such as Console, etc.) may be changed.
- The keyboard data format (7 or 8 bits) may be changed.
- The screen attributes (blink, inverse video, etc.) may be changed on the screen.
- The floppy disk head step rate (speed) can be changed to improve performance.
- The rigid disk may be divided into four disks.

Note: The SET utility is used to change the communications/printer baud rate.

UTILITIES AND PROGRAMS

The following is a list and description of the most commonly used utilities and programs on your CP/M disk.

UTILITY/PROGRAM	DESCRIPTION
BACKUP	Used to backup the data on a rigid disk.
CONFIGUR	Used to modify CP/M for the following particular requirements: • restart command • printer port options • communication port options • I/O device assignments • keyboard data format • screen attributes • floppy disk step rate • rigid disk partitioning
COPY	Used to make an exact copy of a disk.
ED	Used to create and edit files.
FMT	Used to format a rigid disk prior to use.
HELP	Used as a guide for information about CP/M commands, reference manuals, and 820-II special features.
INIT	Used to prepare (initialize) a new disk.
KILLESC	Used to disable the CTRL + ESC command.
PIP	Used to move a file(s) from one disk to another, and to make copies of files.
SET	Used to change the communications/printer port baud rate.
STAT	Used to display the status of a disk; such as disk space and information about the number, size, and kind of files on any given disk.
SWAP	Used to exchange logical disk drive assignments.
SYSGEN	Used to copy the CP/M operating system onto a disk.
TIME	Used to set and display the date and time.
WHATSA	Used to show which logical and physical drives are in use and what type of disks (density/number of sides etc.) are being used.

The following is a list and description of the utilities and programs which are normally used when creating software programs.

UTILITY/PROGRAM	DESCRIPTION
ASM	Used to translate an assembly language source file into a hex file.
DDT	Used to load, alter and test programs written in the CP/M environment.
DUMP	Used to display the contents of a file in hexadecimal.
L80	Used to translate a REL file into a COM file.
LOAD	Used to translate a HEX file into a COM file.
MOVCPM	Used to relocate CP/M for a different memory size.
M80	Used to assemble the 8080 or Z-80 code.
SUBMIT	Used to batch together CP/M commands for automatic processing.
XSUB	Used to input to programs executed in the submit file.

The following is a list of programs used only by the CP/M software and programs. You should not erase these from the disk.

UTILITY/PROGRAM	DESCRIPTION
TERMINAL	Support file for BACKUP utility.
XERBAK	Support file for BACKUP utility.
XERCPY	Support file for BACKUP utility.
XERMAIN	Support file for BACKUP utility.

APPLICATIONS SOFTWARE

The instructions in the 820-II CP/M Operating System Handbook told you how to load CP/M. After loading CP/M, you can run "applications" programs on the 820-II. The applications software may be purchased from XEROX or from other vendors.

When you use your CP/M software to run an applications program for the first time, the program may ask you to define your system. The following information will help you answer these questions:

Your 820-II is configured like a Televideo 950 terminal, or a Lear Siegler ADM-3A display terminal.

Your 820-II has the choice of the following disk drives:

- The 5¼" single sided double density disks have 40 tracks and will have 155K of available space.
- The 5¼" double sided double density disks have 40 tracks per side and will have 322K of available space.
- The 8" single sided double density disks have 77 tracks and have 482K of available space.
- The 8" double sided double density disks have 77 tracks per side and have 980K of available space.
- The rigid disk drive assembly has a 8" double sided floppy drive, and an 8" 10 megabyte rigid drive. The double sided 8" drive is the same as the one above, and the rigid disk has 1,024 tracks and has 8.192 megabytes of available space.
- Always read and follow the instructions that come with the Applications Programs.

Your software is a CP/M 2.2 Operating System.

ROM LEVEL

The Xerox 820-II Basic Operating System monitor is contained in ROM on the CPU board. To check what ROM level is in your system you would turn the 820-II on or press the RESET button if it is already on and read the ROM version level on the screen as shown below.

ROM LEVEL

```
820-II v 0.00 (C) 1982 Xerox Corp

L - Load System
H - Host Terminal
T - Typewriter
```

SOFTWARE SERIAL NUMBER AND LEVEL

To check the software serial number and version level you would load your software and read the screen as shown below:

SOFTWARE LEVEL

SERIAL NUMBER

```
Xerox 60k CP/M vers 2.20 #2-294 DCT0000000
A
```

DIGITAL RESEARCH®

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

AN INTRODUCTION TO CP/M FEATURES AND FACILITIES

REVISION OF JANUARY 1978

Table of Contents

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

pppppppp.*

and

*.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 alphabetics 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), assumming 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 inadvertantly 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:pppppppp.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: = UR1:
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 accidently 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 alphabetics 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),      UC1: (console)
PTR: (reader), UR1: (reader), UR2: (reader)
PTP: (punch),  UP1: (punch),  UP2: (punch)
LPT: (list),   UL1: (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.

PIP PUN:=NUL:,X.ASM,EOF:,NUL: cr — Send 4Ø nulls to the punch device; then copy the X.ASM file to the punch, followed by an end-of-file (ctl-Z) and 4Ø 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 ":ØØ" records in the transfer of Intel hex format file (the I parameter automatically sets the H parameter).

L — Translate upper case alphabetics 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 alphabetics 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 alphabetics 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 [sSUBR1:↑z qJMP L3↑z] cr Copy from the file Y.ASM into the file X.LIB. Start the copy when the string "SUBR1:" 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 accidently 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 aat 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 TØ 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 verskon 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 processsing. 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 Ø through 3 on the MDS, corresponding to CP/M drives A through D, respectively. The CP/M system diskette is inserted into drive Ø, 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 Ø 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.

DIGITAL RESEARCH®

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

CP/M 2 USER'S GUIDE

CP/M 2 USER'S GUIDE

1. AN OVERVIEW OF CP/M 2.0 FACILITIES.

CP/M 2.0 is a high-performance single-console operating system which uses table driven techniques to allow field reconfiguration to match a wide variety of disk capacities. All of the fundamental file restrictions are removed, while maintaining upward compatibility from previous versions of release 1. Features of CP/M 2.0 include field specification of one to sixteen logical drives, each containing up to eight megabytes. Any particular file can reach the full drive size with the capability to expand to thirty-two megabytes in future releases. The directory size can be field configured to contain any reasonable number of entries, and each file is optionally tagged with read/only and system attributes. Users of CP/M 2.0 are physically separated by user numbers, with facilities for file copy operations from one user area to another. Powerful relative-record random access functions are present in CP/M 2.0 which provide direct access to any of the 65536 records of an eight megabyte file.

All disk-dependent portions of CP/M 2.0 are placed into a BIOS-resident "disk parameter block" which is either hand coded or produced automatically using the disk definition macro library provided with CP/M 2.0. The end user need only specify the maximum number of active disks, the starting and ending sector numbers, the data allocation size, the maximum extent of the logical disk, directory size information, and reserved track values. The macros use this information to generate the appropriate tables and table references for use during CP/M 2.0 operation. Deblocking information is also provided which aids in assembly or disassembly of sector sizes which are multiples of the fundamental 128 byte data unit, and the system alteration manual includes general-purpose subroutines which use the this deblocking information to take advantage of larger sector sizes. Use of these subroutines, together with the table driven data access algorithms, make CP/M 2.0 truly a universal data management system.

File expansion is achieved by providing up to 512 logical file extents, where each logical extent contains 16K bytes of data. CP/M 2.0 is structured, however, so that as much as 128K bytes of data is addressed by a single physical extent (corresponding to a single directory entry), thus maintaining compatibility with previous versions while taking full advantage of directory space.

Random access facilities are present in CP/M 2.0 which allow immediate reference to any record of an eight megabyte file. Using CP/M's unique data organization, data blocks are only allocated when actually required and movement to a record position requires little search time. Sequential file access is upward compatible from earlier versions to the full eight megabytes, while random access compatibility stops at 512K byte files. Due to CP/M 2.0's simpler and faster random access, application programmers are encouraged to alter their programs to take full advantage of the 2.0 facilities.

Several CP/M 2.0 modules and utilities have improvements which correspond to the enhanced file system. STAT and PIP both account for file attributes and user areas, while the CCP provides a "login" function to change from one user area to another. The CCP also formats directory displays in a more convenient manner and accounts for both CRT and hard-copy devices in its enhanced line editing functions.

The sections below point out the individual differences between CP/M 1.4 and CP/M 2.0, with the understanding that the reader is either familiar with CP/M 1.4, or has access to the 1.4 manuals. Additional information dealing with CP/M 2.0 I/O system alteration is presented in the Digital Research manual "CP/M 2.0 Alteration Guide."

2. USER INTERFACE.

Console line processing takes CRT-type devices into account with three new control characters, shown with an asterisk in the list below (the symbol "ctl" below indicates that the control key is simultaneously depressed):

```
rub/del removes and echoes last character
 ctl-C  reboot when at beginning of line
 ctl-E  physical end of line
 ctl-H  backspace one character position*
 ctl-J  (line feed) terminates current input*
 ctl-M  (carriage return) terminates input
 ctl-R  retype current line after new line
 ctl-U  remove current line after new line
 ctl-X  backspace to beginning of current line*
```

In particular, note that ctl-H produces the proper backspace overwrite function (ctl-H can be changed internally to another character, such as delete, through a simple single byte change). Further, the line editor keeps track of the current prompt column position so that the operator can properly align data input following a ctl-U, ctl-R, or ctl-X command.

3. CONSOLE COMMAND PROCESSOR (CCP) INTERFACE.

There are four functional differences between CP/M 1.4 and CP/M 2.0 at the console command processor (CCP) level. The CCP now displays directory information across the screen (four elements per line), the USER command is present to allow maintenance of separate files in the same directory, and the actions of the "ERA *.*" and "SAVE" commands have changed. The altered DIR format is self-explanatory, while the USER command takes the form:

USER n

where n is an integer value in the range 0 to 15. Upon cold start, the operator is automatically "logged" into user area number 0, which is compatible with standard CP/M 1.4 directories. The operator may issue the USER command at any time to move to another logical area within the same directory. Drives which are logged-in while addressing one user number are automatically active when the operator moves to another user number since a user number is simply a prefix which accesses particular directory entries on the active disks.

The active user number is maintained until changed by a subsequent USER command, or until a cold start operation when user 0 is again assumed.

Due to the fact that user numbers now tag individual directory entries, the ERA *.* command has a different effect. In version 1.4, this command can be used to erase a directory which has "garbage" information, perhaps resulting from use of a diskette under another operating system (heaven forbid!). In 2.0, however, the ERA *.* command affects only the current user number. Thus, it is necessary to write a simple utility to erase a nonsense disk (the program simply writes the hexadecimal pattern E5 throughout the disk).

The SAVE command in version 1.4 allows only a single memory save operation, with the potential of destroying the memory image due to directory operations following extent boundary changes. Version 2.0, however, does not perform directory operations in user data areas after disk writes, and thus the SAVE operation can be used any number of times without altering the memory image.

4. STAT ENHANCEMENTS.

The STAT program has a number of additional functions which allow disk parameter display, user number display, and file indicator manipulation. The command:

STAT VAL:

produces a summary of the available status commands, resulting in the output:

```
Temp R/O Disk: d:=R/O
Set Indicator: d:filename.typ $R/O $R/W $SYS $DIR
Disk Status  : DSK: d:DSK:
User Status  : USR:
Iobyte Assign:
(list of possible assignments)
```

which gives an instant summary of the possible STAT commands. The command form:

STAT d:filename.typ $S

where "d:" is an optional drive name, and "filename.typ" is an unambiguous or ambiguous file name, produces the output display format:

```
 Size  Recs  Bytes  Ext Acc
   48    48     6k    1 R/O A:ED.COM
   55    55    12k    1 R/O (A:PIP.COM)
65536   128     2k    2 R/W A:X.DAT
```

where the $S parameter causes the "Size" field to be displayed (without the $S, the Size field is skipped, but the remaining fields are displayed). The Size field lists the virtual file size in records, while the "Recs" field sums the number of virtual records in each extent. For files constructed sequentially, the Size and Recs fields are identical. The "Bytes" field lists the actual number of bytes allocated to the corresponding file. The minimum allocation unit is determined at configuration time, and thus the number of bytes corresponds to the record count plus the remaining unused space in the last allocated block for sequential files. Random access files are given data areas only when written, so the Bytes field contains the only accurate allocation figure. In the case of random access, the Size field gives the logical end-of-file record position and the Recs field counts the logical records of each extent (each of these extents, however, may contain unallocated "holes" even though they are added into the record count). The "Ext" field counts the number of logical 16K extents allocated to the file. Unlike version 1.4, the Ext count does not necessarily correspond to the number of directory entries given to the file, since there can be up to 128K bytes (8 logical extents) directly addressed by a single directory entry, depending upon allocation size (in a special case, there are actually 256K bytes which can be directly addressed by a physical extent).

The "Acc" field gives the R/O or R/W access mode, which is changed using the commands shown below. Similarly, the parentheses shown around the PIP.COM file name indicate that it has the "system" indicator set, so that it will not be listed in DIR commands. The four command forms

```
STAT d:filename.typ $R/O
STAT d:filename.typ $R/W
STAT d:filename.typ $SYS
STAT d:filename.typ $DIR
```

set or reset various permanent file indicators. The R/O indicator places the file (or set of files) in a read-only status until changed by a subsequent STAT command. The R/O status is recorded in the directory with the file so that it remains R/O through intervening cold start operations. The R/W indicator places the file in a permanent read/write status. The SYS indicator attaches the system indicator to the file, while the DIR command removes the system indicator. The "filename.typ" may be ambiguous or unambiguous, but in either case, the files whose attributes are changed are listed at the console when the change occurs. The drive name denoted by "d:" is optional.

When a file is marked R/O, subsequent attempts to erase or write into the file result in a terminal BDOS message

```
Bdos Err on d: File R/O
```

The BDOS then waits for a console input before performing a subsequent warm start (a "return" is sufficient to continue). The command form

```
STAT d:DSK:
```

lists the drive characteristics of the disk named by "d:" which is in the range A:, B:, ..., P:. The drive characteristics are listed in the format:

```
      d: Drive Characteristics
  65536: 128 Byte record Capacity
   8192: Kilobyte Drive Capacity
    128: 32  Byte Directory Entries
      0: Checked  Directory Entries
   1024: Records/ Extent
    128: Records/ Block
     58: Sectors/ Track
      2: Reserved Tracks
```

where "d:" is the selected drive, followed by the total record capacity (65536 is an 8 megabyte drive), followed by the total capacity listed in Kilobytes. The directory size is listed next, followed by the "checked" entries. The number of checked entries is usually identical to the directory size for removable media, since this mechanism is used to detect changed media during CP/M operation without an intervening warm start. For fixed media, the number is usually zero, since the media is not changed without at least a cold or warm start. The number of records per extent determines the addressing capacity of each directory entry (1024 times 128 bytes, or

128K in the example above). The number of records per block shows the basic allocation size (in the example, 128 records/block times 128 bytes per record, or 16K bytes per block). The listing is then followed by the number of physical sectors per track and the number of reserved tracks. For logical drives which share the same physical disk, the number of reserved tracks may be quite large, since this mechanism is used to skip lower-numbered disk areas allocated to other logical disks. The command form

STAT DSK:

produces a drive characteristics table for all currently active drives. The final STAT command form is

STAT USR:

which produces a list of the user numbers which have files on the currently addressed disk. The display format is:

```
Active User : 0
Active Files: 0 1 3
```

where the first line lists the currently addressed user number, as set by the last CCP USER command, followed by a list of user numbers scanned from the current directory. In the above case, the active user number is 0 (default at cold start), with three user numbers which have active files on the current disk. The operator can subsequently examine the directories of the other user numbers by logging-in with USER 1, USER 2, or USER 3 commands, followed by a DIR command at the CCP level.

5. PIP ENHANCEMENTS.

PIP provides three new functions which account for the features of CP/M 2.0. All three functions take the form of file parameters which are enclosed in square brackets following the appropriate file names. The commands are:

Gn	Get File from User number n (n in the range 0 - 15)
W	Write over R/O files without console interrogation
R	Read system files

The G command allows one user area to receive data files from another. Assuming the operator has issued the USER 4 command at the CCP level, the PIP statement

```
PIP X.Y = X.Y[G2]
```

reads file X.Y from user number 2 into user area number 4. The command

```
PIP A:=A:*.*[G2]
```

copies all of the files from the A drive directory for user number 2 into the A drive directory of the currently logged user number. Note that to ensure file security, one cannot copy files into a different area than the one which is currently addressed by the USER command.

Note also that the PIP program itself is initially copied to a user area (so that subsequent files can be copied) using the SAVE command. The sequence of operations shown below effectively moves PIP from one user area to the next.

```
USER 0              login user 0
DDT PIP.COM         load PIP to memory
(note PIP size s)
G0                  return to CCP
USER 3              login user 3
SAVE s PIP.COM
```

where s is the integral number of memory "pages" (256 byte segments) occupied by PIP. The number s can be determined when PIP.COM is loaded under DDT, by referring to the value under the "NEXT" display. If for example, the next available address is 1D00, then PIP.COM requires 1C hexadecimal pages (or 1 times 16 + 12 = 28 pages), and thus the value of s is 28 in the subsequent save. Once PIP is copied in this manner, it can then be copied to another disk belonging to the same user number through normal pip transfers.

Under normal operation, PIP will not overwrite a file which is set to a permanent R/O status. If attempt is made to overwrite a R/O file, the prompt

DESTINATION FILE IS R/O, DELETE (Y/N)?

is issued. If the operator responds with the character "y" then the file is overwritten. Otherwise, the response

** NOT DELETED **

is issued, the file transfer is skippped, and PIP continues with the next operation in sequence. In order to avoid the prompt and response in the case of R/O file overwrite, the command line can include the W parameter, as shown below

PIP A:=B:*.COM[W]

which copies all non-system files to the A drive from the B drive, and overwrites any R/O files in the process. If the operation involves several concatenated files, the W parameter need only be included with the last file in the list, as shown in the following example

PIP A.DAT = B.DAT,F:NEW.DAT,G:OLD.DAT[W]

Files with the system attribute can be included in PIP transfers if the R parameter is included, otherwise system files are not recognized. The command line

PIP ED.COM = B:ED.COM[R]

for example, reads the ED.COM file from the B drive, even if it has been marked as a R/O and system file. The system file attributes are copied, if present.

It should be noted that downward compatibility with previous versions of CP/M is only maintained if the file does not exceed one megabyte, no file attributes are set, and the file is created by user 0. If compatibility is required with non-standard (e.g., "double density") versions of 1.4, it may be necessary to select 1.4 compatibility mode when constructing the internal disk parameter block (see the "CP/M 2.0 Alteration Guide," and refer to Section 10 which describes BIOS differences).

6. ED ENHANCEMENTS.

The CP/M standard program editor provides several new facilities in the 2.Ø release. Experience has shown that most operators use the relative line numbering feature of ED, and thus the editor has the "v" (Verify Line) option set as an initial value. The operator can, of course, disable line numbering by typing the "-v" command. If you are not familiar with the ED line number mode, you may wish to refer to the Appendix in the ED user's guide, where the "v" command is described.

ED also takes file attributes into account. If the operator attempts to edit a read/only file, the message

** FILE IS READ/ONLY **

appears at the console. The file can be loaded and examined, but cannot be altered in any way. Normally, the operator simply ends the edit session, and uses STAT to change the file attribute to R/W. If the edited file has the "system" attribute set, the message

"SYSTEM" FILE NOT ACCESSIBLE

is displayed at the console, and the edit session is aborted. Again, the STAT program can be used to change the system attribute, if desired.

Finally, the insert mode ("i") command allows CRT line editing functions, as described in Section 2, above.

7. THE XSUB FUNCTION.

An additional utility program is supplied with version 2.0 of CP/M, called XSUB, which extends the power of the SUBMIT facility to include line input to programs as well as the console command processor. The XSUB command is included as the first line of your submit file and, when executed, self-relocates directly below the CCP. All subsequent submit command lines are processed by XSUB, so that programs which read buffered console input (BDOS function 10) receive their input directly from the submit file. For example, the file SAVER.SUB could contain the submit lines:

```
XSUB
DDT
I$1.HEX
R
G0
SAVE 1 $2.COM
```

with a subsequent SUBMIT command:

```
SUBMIT SAVER X Y
```

which substitutes X for $1 and Y for $2 in the command stream. The XSUB program loads, followed by DDT which is sent the command lines "IX.HEX" "R" and "G0" thus returning to the CCP. The final command "SAVE 1 Y.COM" is processed by the CCP.

The XSUB program remains in memory, and prints the message

```
(xsub active)
```

on each warm start operation to indicate its presence. Subsequent submit command streams do not require the XSUB, unless an intervening cold start has occurred. Note that XSUB must be loaded after DESPOOL, if both are to run simultaneously.

8. BDOS INTERFACE CONVENTIONS.

CP/M 2.0 system calls take place in exactly the same manner as earlier versions, with a call to location 0005H, function number in register C, and information address in register pair DE. Single byte values are returned in register A, with double byte values returned in HL (for reasons of compatibility, register A = L and register B = H upon return in all cases). A list of CP/M 2.0 calls is given below, with an asterisk following functions which are either new or revised from version 1.4 to 2.0. Note that a zero value is returned for out-of range function numbers.

0	System Reset	19*	Delete File
1	Console Input	20	Read Sequential
2	Console Output	21	Write Sequential
3	Reader Input	22*	Make File
4	Punch Output	23*	Rename File
5	List Output	24*	Return Login Vector
6*	Direct Console I/O	25	Return Current Disk
7	Get I/O Byte	26	Set DMA Address
8	Set I/O Byte	27	Get Addr(Alloc)
9	Print String	28*	Write Protect Disk
10*	Read Console Buffer	29*	Get Addr(R/O Vector)
11	Get Console Status	30*	Set File Attributes
12*	Return Version Number	31*	Get Addr(Disk Parms)
13	Reset Disk System	32*	Set/Get User Code
14	Select Disk	33*	Read Random
15*	Open File	34*	Write Random
16	Close File	35*	Compute File Size
17*	Search for First	36*	Set Random Record
18*	Search for Next		

(Functions 28, 29, and 32 should be avoided in application programs to maintain upward compatibility with MP/M.) The new or revised functions are described below.

Function 6: Direct Console I/O.

Direct Console I/O is supported under CP/M 2.0 for those applications where it is necessary to avoid the BDOS console I/O operations. Programs which currently perform direct I/O through the BIOS should be changed to use direct I/O under BDOS so that they can be fully supported under future releases of MP/M and CP/M.

Upon entry to function 6, register E either contains hexadecimal FF, denoting a console input request, or register E contains an ASCII character. If the input value is FF, then function 6 returns A = 00 if no character is ready, otherwise A contains the next console input character.

If the input value in E is not FF, then function 6 assumes that E contains a valid ASCII character which is sent to the console.

Function 10: Read Console Buffer.

The console buffer read operation remains unchanged except that console line editing is supported, as described in Section 2. Note also that certain functions which return the carriage to the leftmost position (e.g., ctl-X) do so only to the column position where the prompt ended (previously, the carriage returned to the extreme left margin). This new convention makes operator data input and line correction more legible.

Function 12: Return Version Number.

Function 12 has been redefined to provide information which allows version-independent programming (this was previously the "lift head" function which returned HL=0000 in version 1.4, but performed no operation). The value returned by function 12 is a two-byte value, with H = 00 for the CP/M release (H = 01 for MP/M), and L = 00 for all releases previous to 2.0. CP/M 2.0 returns a hexadecimal 20 in register L, with subsequent version 2 releases in the hexadecimal range 21, 22, through 2F. Using function 12, for example, you can write application programs which provide both sequential and random access functions, with random access disabled when operating under early releases of CP/M.

In the file operations described below, DE addresses a file control block (FCB). Further, all directory operations take place in a reserved area which does not affect write buffers as was the case in version 1.4, with the exception of Search First and Search Next, where compatibility is required.

The File Control Block (FCB) data area consists of a sequence of 33 bytes for sequential access, and a series of 36 bytes in the case that the file is accessed randomly. The default file control block normally located at 005CH can be used for random access files, since bytes 007DH, 007EH, and 007FH are available for this purpose. For notational purposes, the FCB format is shown with the following fields:

```
-------------------------------------------------------------
|dr|f1|f2|/ /|f8|t1|t2|t3|ex|s1|s2|rc|d0|/ /|dn|cr|r0|r1|r2|
-------------------------------------------------------------
 00 01 02 ... 08 09 10 11 12 13 14 15 16 ... 31 32 33 34 35
```

where

```
dr          drive code (0 - 16)
            0 => use default drive for file
            1 => auto disk select drive A,
            2 => auto disk select drive B,
            ...
            16=> auto disk select drive P.

f1...f8     contain the file name in ASCII
            upper case, with high bit = 0

t1,t2,t3    contain the file type in ASCII
            upper case, with high bit = 0
            t1', t2', and t3' denote the
            bit of these positions,
            t1' = 1 => Read/Only file,
            t2' = 1 => SYS file, no DIR list

 ex         contains the current extent number,
            normally set to 00 by the user, but
            in range 0 - 31 during file I/O

 s1         reserved for internal system use

 s2         reserved for internal system use, set
            to zero on call to OPEN, MAKE, SEARCH

 rc         record count for extent "ex,"
            takes on values from 0 - 128

 d0...dn    filled-in by CP/M, reserved for
            system use

 cr         current record to read or write in
            a sequential file operation, normally
            set to zero by user

 r0,r1,r2   optional random record number in the
            range 0-65535, with overflow to r2,
            r0,r1 constitute a 16-bit value with
            low byte r0, and high byte r1
```

Function 15: Open File.

The Open File operation is identical to previous definitions, with the exception that byte s2 is automatically zeroed. Note that previous versions of CP/M defined this byte as zero, but made no checks to assure compliance. Thus, the byte is cleared to ensure upward compatibility with the latest version, where it is required.

Function 17: Search for First.

Search First scans the directory for a match with the file given by the FCB addressed by DE. The value 255 (hexadecimal FF) is returned if the file is not found, otherwise a value of A equal to 0, 1, 2, or 3 is returned indicating the file is present. In the case that the file is found, the current DMA address is filled with the record containing the directory entry, and the relative starting position is A * 32 (i.e., rotate the A register left 5 bits, or ADD A five times). Although not normally required for application programs, the directory information can be extracted from the buffer at this position.

An ASCII question mark (63 decimal, 3F hexadecimal) in any position from f1 through ex matches the corresponding field of any directory entry on the default or auto-selected disk drive. If the dr field contains an ASCII question mark, then the auto disk select function is disabled, the default disk is searched, with the search function returning any matched entry, allocated or free, belonging to any user number. This latter function is not normally used by application programs, but does allow complete flexibility to scan all current directory values. If the dr field is not a question mark, the s2 byte is automatically zeroed.

Function 18: Search for Next.

The Search Next function is similar to the Search First function, except that the directory scan continues from the last matched entry. Similar to function 17, function 18 returns the decimal value 255 in A when no more directory items match.

Function 19: Delete File.

The Delete File function removes files which match the FCB addressed by DE. The filename and type may contain ambiguous references (i.e., question marks in various positions), but the drive select code cannot be ambiguous, as in the Search and Search Next functions.

Function 19 returns a decimal 255 if the reference file or files could not be found, otherwise a value in the range 0 to 3 is returned.

Function 22: Make File.

The Make File operation is identical to previous versions of CP/M, except that byte s2 is zeroed upon entry to the BDOS.

Function 23: Rename File.

The Actions of the file rename functions are the same as previous releases except that the value 255 is returned if the rename function is unsuccessful (the file to rename could not be found), otherwise a value in the range Ø to 3 is returned.

Function 24: Return Login Vector.

The login vector value returned by CP/M 2.Ø is a 16-bit value in HL, where the least significant bit of L corresponds to the first drive A, and the high order bit of H corresponds to the sixteenth drive, labelled P. Note that compatibility is maintained with earlier releases, since registers A and L contain the same values upon return.

Function 28: Write Protect Current Disk.

The disk write protect function provides temporary write protection for the currently selected disk. Any attempt to write to the disk, before the next cold or warm start operation produces the message

Bdos Err on d: R/O

Function 29: Get R/O Vector.

Function 29 returns a bit vector in register pair HL which indicates drives which have the temporary read/only bit set. Similar to function 24, the least significant bit corresponds to drive A, while the most significant bit corresponds to drive P. The R/O bit is set either by an explicit call to function 28, or by the automatic software mechanisms within CP/M which detect changed disks.

Function 3Ø: Set File Attributes.

The Set File Attributes function allows programmatic manipulation of permanent indicators attached to files. In particular, the R/O and System attributes (t1' and t2' above) can be set or reset. The DE pair addresses an unambiguous file name with the appropriate attributes set or reset. Function 3Ø searches for a match, and changes the matched directory entry to contain the selected indicators. Indicators f1' through f4' are not presently used, but may be useful for applications programs, since they are not involved in the matching process during file open and close operations. Indicators f5' through f8' and t3' are reserved for future system expansion.

Function 31: Get Disk Parameter Block Address.

The address of the BIOS resident disk parameter block is returned in HL as a result of this function call. This address can be used for either of two purposes. First, the disk parameter values can be extracted for display and space computation purposes, or transient programs can dynamically change the values of current disk parameters when the disk environment changes, if required. Normally, application programs will not require this facility.

Function 32: Set or Get User Code.

An application program can change or interrogate the currently active user number by calling function 32. If register E = FF hexadecimal, then the value of the current user number is returned in register A, where the value is in the range Ø to 31. If register E is not FF, then the current user number is changed to the value of E (modulo 32).

Function 33: Read Random.

The Read Random function is similar to the sequential file read operation of previous releases, except that the read operation takes place at a particular record number, selected by the 24-bit value constructed from the three byte field following the FCB (byte positions rØ at 33, r1 at 34, and r2 at 35). Note that the sequence of 24 bits is stored with least significant byte first (rØ), middle byte next (r1), and high byte last (r2). CP/M release 2.Ø does not reference byte r2, except in computing the size of a file (function 35). Byte r2 must be zero, however, since a non-zero value indicates overflow past the end of file.

Thus, in version 2.Ø, the rØ,r1 byte pair is treated as a double-byte, or "word" value, which contains the record to read. This value ranges from Ø to 65535, providing access to any particular record of the 8 megabyte file. In order to process a file using random access, the base extent (extent Ø) must first be opened. Although the base extent may or may not contain any allocated data, this ensures that the file is properly recorded in the directory, and is visible in DIR requests. The selected record number is then stored into the random record field (rØ,r1), and the BDOS is called to read the record. Upon return from the call, register A either contains an error code, as listed below, or the value 00 indicating the operation was successful. In the latter case, the current DMA address contains the randomly accessed record. Note that contrary to the sequential read operation, the record number is not advanced. Thus, subsequent random read operations continue to read the same record.

Upon each random read operation, the logical extent and current record values are automatically set. Thus, the file can be sequentially read or written, starting from the current randomly accessed position. Note, however, that in this case, the last randomly read record will be re-read as you switch from random mode to sequential read, and the last record will be re-written as you switch to a sequential write operation. You can, of course, simply advance the random record position following each random read or write to obtain the effect of a sequential I/O operation.

Error codes returned in register A following a random read are listed below.

- 01 reading unwritten data
- 02 (not returned in random mode)
- 03 cannot close current extent
- 04 seek to unwritten extent
- 05 (not returned in read mode)
- 06 seek past physical end of disk

Error code 01 and 04 occur when a random read operation accesses a data block which has not been previously written, or an extent which has not been created, which are equivalent conditions. Error 3 does not normally occur under proper system operation, but can be cleared by simply re-reading, or re-opening extent zero as long as the disk is not physically write protected. Error code 06 occurs whenever byte r2 is non-zero under the current 2.0 release. Normally, non-zero return codes can be treated as missing data, with zero return codes indicating operation complete.

Function 34: Write Random.

The Write Random operation is initiated similar to the Read Random call, except that data is written to the disk from the current DMA address. Further, if the disk extent or data block which is the target of the write has not yet been allocated, the allocation is performed before the write operation continues. As in the Read Random operation, the random record number is not changed as a result of the write. The logical extent number and current record positions of the file control block are set to correspond to the random record which is being written. Again, sequential read or write operations can commence following a random write, with the notation that the currently addressed record is either read or rewritten again as the sequential operation begins. You can also simply advance the random record position following each write to get the effect of a sequential write operation. Note that in particular, reading or writing the last record of an extent in random mode does not cause an automatic extent switch as it does in sequential mode under either CP/M 1.4 or CP/M 2.0.

The error codes returned by a random write are identical to the random read operation with the addition of error code 05, which indicates that a new extent cannot be created due to directory overflow.

Function 35: Compute File Size.

When computing the size of a file, the DE register pair addresses an FCB in random mode format (bytes r0, r1, and r2 are present). The FCB contains an unambiguous file name which is used in the directory scan. Upon return, the random record bytes contain the "virtual" file size which is, in effect, the record address of the record following the end of the file. if, following a call to function 35, the high record byte r2 is 01, then the file contains the maximum record count 65536 in version 2.0. Otherwise, bytes r0 and r1 constitute a 16-bit value (r0 is the least significant byte, as before) which is the file size.

Data can be appended to the end of an existing file by simply calling function 35 to set the random record position to the end of file, then performing a sequence of random writes starting at the preset record address.

The virtual size of a file corresponds to the physical size when the file is written sequentially. If, instead, the file was created in random mode and "holes" exist in the allocation, then the file may in fact contain fewer records than the size indicates. If, for example, only the last record of an eight megabyte file is written in random mode (i.e., record number 65535), then the virtual size is 65536 records, although only one block of data is actually allocated.

Function 36: Set Random Record.

The Set Random Record function causes the BDOS to automatically produce the random record position from a file which has been read or written sequentially to a particular point. The function can be useful in two ways.

First, it is often necessary to initially read and scan a sequential file to extract the positions of various "key" fields. As each key is encountered, function 36 is called to compute the random record position for the data corresponding to this key. If the data unit size is 128 bytes, the resulting record position is placed into a table with the key for later retrieval. After scanning the entire file and tabularizing the keys and their record numbers, you can move instantly to a particular keyed record by performing a random read using the corresponding random record number which was saved earlier. The scheme is easily generalized when variable record lengths are involved since the program need only store the buffer-relative byte position along with the key and record number in order to find the exact starting position of the keyed data at a later time.

A second use of function 36 occurs when switching from a sequential read or write over to random read or write. A file is sequentially accessed to a particular point in the file, function 36 is called which sets the record number, and subsequent random read and write operations continue from the selected point in the file.

This section is concluded with a rather extensive, but complete example of random access operation. The program listed below performs the simple function of reading or writing random records upon command from the terminal. Given that the program has been created, assembled, and placed into a file labelled RANDOM.COM, the CCP level command:

RANDOM X.DAT

starts the test program. The program looks for a file by the name X.DAT (in this particular case) and, if found, proceeds to prompt the console for input. If not found, the file is created before the prompt is given. Each prompt takes the form

next command?

and is followed by operator input, terminated by a carriage return. The input commands take the form

nW nR Q

where n is an integer value in the range 0 to 65535, and W, R, and Q are simple command characters corresponding to random write, random read, and quit processing, respectively. If the W command is issued, the RANDOM program issues the prompt

type data:

The operator then responds by typing up to 127 characters, followed by a carriage return. RANDOM then writes the character string into the X.DAT file at record n. If the R command is issued, RANDOM reads record number n and displays the string value at the console. If the Q command is issued, the X.DAT file is closed, and the program returns to the console command processor. In the interest of brevity (ok, so the program's not so brief), the only error message is

error, try again

The program begins with an initialization section where the input file is opened or created, followed by a continuous loop at the label "ready" where the individual commands are interpreted. The default file control block at 005CH and the default buffer at 0080H are used in all disk operations. The utility subroutines then follow, which contain the principal input line processor, called "readc." This particular program shows the elements of random access processing, and can be used as the basis for further program development.

```
                ;*************************************************
                ;*                                               *
                ;* sample random access program for cp/m 2.0     *
                ;*                                               *
                ;*************************************************
0100                    org     100h    ;base of tpa
                ;
0000 =          reboot  equ     0000h   ;system reboot
0005 =          bdos    equ     0005h   ;bdos entry point
                ;
0001 =          coninp  equ     1       ;console input function
0002 =          conout  equ     2       ;console output function
0009 =          pstring equ     9       ;print string until '$'
000a =          rstring equ     10      ;read console buffer
000c =          version equ     12      ;return version number
000f =          openf   equ     15      ;file open function
0010 =          closef  equ     16      ;close function
0016 =          makef   equ     22      ;make file function
0021 =          readr   equ     33      ;read random
0022 =          writer  equ     34      ;write random
                ;
005c =          fcb     equ     005ch   ;default file control block
007d =          ranrec  equ     fcb+33  ;random record position
007f =          ranovf  equ     fcb+35  ;high order (overflow) byte
0080 =          buff    equ     0080h   ;buffer address
                ;
000d =          cr      equ     0dh     ;carriage return
000a =          lf      equ     0ah     ;line feed
                ;
                ;*************************************************
                ;*                                               *
                ;* load SP, set-up file for random access        *
                ;*                                               *
                ;*************************************************
0100 31bc0              lxi     sp,stack
                ;
                ;       version 2.0?
0103 0e0c               mvi     c,version
0105 cd050              call    bdos
0108 fe20               cpi     20h     ;version 2.0 or better?
010a d2160              jnc     versok
                ;       bad version, message and go back
010d 111b0              lxi     d,badver
0110 cdda0              call    print
0113 c3000              jmp     reboot
                ;
                versok:
                ;       correct version for random access
```

```
0116 0e0f              mvi     c,openf ;open default fcb
0118 115c00            lxi     d,fcb
011b cd0500            call    bdos
011e 3c                inr     a       ;err 255 becomes zero
011f c23701            jnz     ready
                ;
                ;       cannot open file, so create it
0122 0e16              mvi     c,makef
0124 115c00            lxi     d,fcb
0127 cd0500            call    bdos
012a 3c                inr     a       ;err 255 becomes zero
012b c23701            jnz     ready
                ;
                ;       cannot create file, directory full
012e 113a00            lxi     d,nospace
0131 cdda00            call    print
0134 c30000            jmp     reboot  ;back to ccp
                ;
                ;***************************************************
                ;*                                                 *
                ;*  loop back to "ready" after each command        *
                ;*                                                 *
                ;***************************************************
                ;
                ready:
                ;       file is ready for processing
                ;
0137 cde500            call    readcom ;read next command
013a 227d00            shld    ranrec  ;store input record#
013d 217f00            lxi     h,ranovf
0140 3600              mvi     m,0     ;clear high byte if set
0142 fe51              cpi     'Q'     ;quit?
0144 c25601            jnz     notq
                ;
                ;       quit processing, close file
0147 0e10              mvi     c,closef
0149 115c00            lxi     d,fcb
014c cd0500            call    bdos
014f 3c                inr     a       ;err 255 becomes 0
0150 cab900            jz      error   ;error message, retry
0153 c30000            jmp     reboot  ;back to ccp
                ;
                ;***************************************************
                ;*                                                 *
                ;* end of quit command, process write              *
                ;*                                                 *
                ;***************************************************
                notq:
                ;       not the quit command, random write?
0156 fe57              cpi     'W'
0158 c28901            jnz     notw
                ;
                ;       this is a random write, fill buffer until cr
015b 114d00            lxi     d,datmsg
015e cdda00            call    print   ;data prompt
```

```
0161 0e7f          mvi     c,127   ;up to 127 characters
0163 21800         lxi     h,buff  ;destination
            rloop: ;read next character to buff
0166 c5            push    b       ;save counter
0167 e5            push    h       ;next destination
0168 cdc20         call    getchr  ;character to a
016b e1            pop     h       ;restore counter
016c c1            pop     b       ;restore next to fill
016d fe0d          cpi     cr      ;end of line?
016f ca780         jz      erloop
            ;      not end, store character
0172 77            mov     m,a
0173 23            inx     h       ;next to fill
0174 0d            dcr     c       ;counter goes down
0175 c2660         jnz     rloop   ;end of buffer?
            erloop:
            ;      end of read loop, store 00
0178 3600          mvi     m,0
            ;
            ;      write the record to selected record number
017a 0e22          mvi     c,writer
017c 115c0         lxi     d,fcb
017f cd050         call    bdos
0182 b7            ora     a       ;error code zero?
0183 c2b90         jnz     error   ;message if not
0186 c3370         jmp     ready   ;for another record
            ;
            ;*******************************************************
            ;*                                                     *
            ;* end of write command, process read                  *
            ;*                                                     *
            ;*******************************************************
            notw:
            ;      not a write command, read record?
0189 fe52          cpi     'R'
018b c2b90         jnz     error   ;skip if not
            ;
            ;      read random record
018e 0e21          mvi     c,readr
0190 115c0         lxi     d,fcb
0193 cd050         call    bdos
0196 b7            ora     a       ;return code 00?
0197 c2b90         jnz     error
            ;
            ;      read was successful, write to console
019a cdcf0         call    crlf    ;new line
019d 0e80          mvi     c,128   ;max 128 characters
019f 21800         lxi     h,buff  ;next to get
            wloop:
01a2 7e            mov     a,m     ;next character
01a3 23            inx     h       ;next to get
01a4 e67f          ani     7fh     ;mask parity
01a6 ca370         jz      ready   ;for another command if 00
01a9 c5            push    b       ;save counter
01aa e5            push    h       ;save next to get
```

```
01ab fe20           cpi                     ;graphic?
01ad d4c80          cnc         putchr      ;skip output if not
01b0 e1             pop         h
01b1 c1             pop         b
01b2 0d             dcr         c           ;count=count-1
01b3 c2a20          jnz         wloop
01b6 c3370          jmp         ready
              ;
              ;**************************************************
              ;*                                                *
              ;* end of read command, all errors end-up here    *
              ;*                                                *
              ;**************************************************
              ;
              error:
01b9 11590          lxi         d,errmsg
01bc cdda0          call        print
01bf c3370          jmp         ready
              ;
              ;**************************************************
              ;*                                                *
              ;* utility subroutines for console i/o            *
              ;*                                                *
              ;**************************************************
              getchr:
                    ;read next console character to a
01c2 0e01           mvi         c,coninp
01c4 cd050          call        bdos
01c7 c9             ret
              ;
              putchr:
                    ;write character from a to console
01c8 0e02           mvi         c,conout
01ca 5f             mov         e,a         ;character to send
01cb cd050          call        bdos        ;send character
01ce c9             ret
              ;
              crlf:
                    ;send carriage return line feed
01cf 3e0d           mvi         a,cr        ;carriage return
01d1 cdc80          call        putchr
01d4 3e0a           mvi         a,lf        ;line feed
01d6 cdc80          call        putchr
01d9 c9             ret
              ;
              print:
                    ;print the buffer addressed by de until $
01da d5             push        d
01db cdcf0          call        crlf
01de d1             pop         d           ;new line
01df 0e09           mvi         c,pstring
01e1 cd050          call        bdos        ;print the string
01e4 c9             ret
              ;
              readcom:
```

```
                        ;read the next command line to the conbuf
0le5 116b0              lxi     d,prompt
0le8 cdda0              call    print   ;command?
0leb 0e0a               mvi     c,rstring
0led 117a0              lxi     d,conbuf
0lf0 cd050              call    bdos    ;read command line
                ;       command line is present, scan it
0lf3 21000              lxi     h,0     ;start with 0000
0lf6 117c0              lxi     d,conlin;command line
0lf9 la         readc:  ldax    d       ;next command character
0lfa 13                 inx     d       ;to next command position
0lfb b7                 ora     a       ;cannot be end of command
0lfc c8                 rz
                ;       not zero, numeric?
0lfd d630               sui     '0'
0lff fe0a               cpi     10      ;carry if numeric
0201 d2130              jnc     endrd
                ;       add-in next digit
0204 29                 dad     h       ;*2
0205 4d                 mov     c,l
0206 44                 mov     b,h     ;bc = value * 2
0207 29                 dad     h       ;*4
0208 29                 dad     h       ;*8
0209 09                 dad     b       ;*2 + *8 = *10
020a 85                 add     l       ;+digit
020b 6f                 mov     l,a
020c d2f90              jnc     readc   ;for another char
020f 24                 inr     h       ;overflow
0210 c3f90              jmp     readc   ;for another char
                endrd:
                ;       end of read, restore value in a
0213 c630               adi     '0'     ;command
0215 fe61               cpi     'a'     ;translate case?
0217 d8                 rc
                ;       lower case, mask lower case bits
0218 e65f               ani     101$1111b
021a c9                 ret
                ;
                ;*****************************************************
                ;*                                                   *
                ;*  string data area for console messages            *
                ;*                                                   *
                ;*****************************************************
                badver:
021b 536f79             db      'sorry, you need cp/m version 2$'
                nospace:
023a 4e6f29             db      'no directory space$'
                datmsg:
024d 547970             db      'type data: $'
                errmsg:
0259 457272             db      'error, try again.$'
                prompt:
026b 4e6570             db      'next command? $'
                ;
```

```
                ;*************************************************
                ;*                                               *
                ;*  fixed and variable data area                 *
                ;*                                               *
                ;*************************************************
Ø27a 21         conbuf: db      conlen  ;length of console buffer
Ø27b            consiz: ds      1       ;resulting size after read
Ø27c            conlin: ds      32      ;length 32 buffer
ØØ21 =          conlen  equ     $-consiz
                ;
Ø29c                    ds      32      ;16 level stack
                stack:
Ø2bc                    end
```

9. CP/M 2.0 MEMORY ORGANIZATION.

Similar to earlier versions, CP/M 2.0 is field-altered to fit various memory sizes, depending upon the host computer memory configuration. Typical base addresses for popular memory sizes are shown in the table below.

Module	20k	24k	32k	48k	64k
CCP	3400H	4400H	6400H	A400H	E400H
BDOS	3C00H	4C00H	6C00H	AC00H	EC00H
BIOS	4A00H	5A00H	7A00H	BA00H	FA00H
Top of Ram	4FFFH	5FFFH	7FFFH	BFFFH	FFFFH

The distribution disk contains a CP/M 2.0 system configured for a 20k Intel MDS-800 with standard IBM 8" floppy disk drives. The disk layout is shown below:

Sector	Track 00	Module	Track 01	Module
1	(Bootstrap Loader)		4080H	BDOS + 480H
2	3400H	CCP + 000H	4100H	BDOS + 500H
3	3480H	CCP + 080H	4180H	BDOS + 580H
4	3500H	CCP + 100H	4200H	BDOS + 600H
5	3580H	CCP + 180H	4280H	BDOS + 680H
6	3600H	CCP + 200H	4300H	BDOS + 700H
7	3680H	CCP + 280H	4380H	BDOS + 780H
8	3700H	CCP + 300H	4400H	BDOS + 800H
9	3780H	CCP + 380H	4480H	BDOS + 880H
10	3800H	CCP + 400H	4500H	BDOS + 900H
11	3880H	CCP + 480H	4580H	BDOS + 980H
12	3900H	CCP + 500H	4600H	BDOS + A00H
13	3980H	CCP + 580H	4680H	BDOS + A80H
14	3A00H	CCP + 600H	4700H	BDOS + B00H
15	3A80H	CCP + 680H	4780H	BDOS + B80H
16	3B00H	CCP + 700H	4800H	BDOS + C00H
17	3B80H	CCP + 780H	4880H	BDOS + C80H
18	3C00H	BDOS + 000H	4900H	BDOS + D00H
19	3C80H	BDOS + 080H	4980H	BDOS + D80H
20	3D00H	BDOS + 100H	4A00H	BIOS + 000H
21	3D80H	BDOS + 180H	4A80H	BIOS + 080H
22	3E00H	BDOS + 200H	4B00H	BIOS + 100H
23	3E80H	BDOS + 280H	4B80H	BIOS + 180H
24	3F00H	BDOS + 300H	4C00H	BIOS + 200H
25	3F80H	BDOS + 380H	4C80H	BIOS + 280H
26	4000H	BDOS + 400H	4D00H	BIOS + 300H

In particular, note that the CCP is at the same position on the disk, and occupies the same space as version 1.4. The BDOS portion, however, occupies one more 256-byte page and the BIOS portion extends through the remainder of track 01. Thus, the CCP is 800H (2048 decimal) bytes in length, the BDOS is E00H (3584 decimal) bytes in length, and the BIOS is up to 380H (898 decimal) bytes in length. In version 2.0, the BIOS portion contains the standard subroutines of 1.4, along with some initialized table space, as described in the following section.

10. BIOS DIFFERENCES.

The CP/M 2.0 Basic I/O System differs only slightly in concept from its predecesssors. Two new jump vector entry points are defined, a new sector translation subroutine is included, and a disk characteristics table must be defined. The skeletal form of these changes are found in the program shown below.

```
                org     4000h
                maclib  diskdef
                jmp     boot
;               ...
                jmp     listst  ;list status
                jmp     sectran ;sector translate
                disks   4
;               large capacity drive
bpb             equ     16*1024 ;bytes per block
rpb             equ     bpb/128 ;records per block
maxb            equ     65535/rpb ;max block number
                diskdef 0,1,58,3,bpb,maxb+1,128,0,2
                diskdef 1,1,58,,bpb,maxb+1,128,0,2
                diskdef 2,0
                diskdef 3,1
;
boot:           ret     ;nop
;
listst:         xra     a       ;nop
                ret
;
seldsk:
                ;drive number in c
                lxi     h,0     ;0000 in hl produces select error
                mov     a,c     ;a is disk number 0 ... ndisks-1
                cpi     ndisks  ;less than ndisks?
                rnc             ;return with HL = 0000 if not
;               proper disk number, return dpb element address
                mov     l,c
                dad     h       ;*2
                dad     h       ;*4
                dad     h       ;*8
                dad     h       ;*16
                lxi     d,dpbase
                dad     d       ;HL=.dpb
                ret
;
selsec:
                ;sector number in c
                lxi     h,sector
                mov     m,c
                ret
;
sectran:
                ;translate sector BC using table at DE
                xchg            ;HL = .tran
                dad     b       ;single precision tran
```

```
48: ;           dad b again if double precision tran
49:             mov     l,m      ;only low byte necessary here
50: ;           fill both H and L if double precision tran
51:             ret              ;HL = ??ss
52: ;
53: sector:     ds      1
54:             endef
55:             end
```

Referring to the program shown above, lines 3-6 represent the BIOS entry vector of 17 elements (version 1.4 defines only 15 jump vector elements). The last two elements provide access to the "LISTST" (List Status) entry point for DESPOOL. The use of this particular entry point is defined in the DESPOOL documentation, and is no different than the previous 1.4 release. It should be noted that the 1.4 DESPOOL program will not operate under version 2.0, but an update version will be available from Digital Research in the near future.

The "SECTRAN" (Sector Number Translate) entry shown in the jump vector at line 6 provides access to a BIOS-resident sector translation subroutine. This mechanism allows the user to specify the sector skew factor and translation for a particular disk system, and is described below.

A macro library is shown in the listing, called DISKDEF, included on line 2, and referenced in 12-15. Although it is not necessary to use the macro library, it greatly simplifies the disk definition process. You must have access to the MAC macro assembler, of course, to use the DISKDEF facility, while the macro library is included with all CP/M 2.0 distribution disks. (See the CP/M 2.0 Alteration Guide for formulas which you can use to hand-code the tables produced by the DISKDEF library).

A BIOS disk definition consists of the following sequence of macro statements:

```
MACLIB    DISKDEF
......
DISKS     n
DISKDEF   0,...
DISKDEF   1,...
......
DISKDEF   n-1
......
ENDEF
```

where the MACLIB statement loads the DISKDEF.LIB file (on the same disk as your BIOS) into MAC's internal tables. The DISKS macro call follows, which specifies the number of drives to be configured with your system, where n is an integer in the range 1 to 16. A series of DISKDEF macro calls then follow which define the characteristics of each logical disk, 0 through n-1 (corresponding to logical drives A through P). Note that the DISKS and DISKDEF macros generate in-line fixed data tables, and thus must be placed in a non-executable portion of your BIOS, typically directly following the BIOS jump vector.

The remaining portion of your BIOS is defined following the DISKDEF macros, with the ENDEF macro call immediately preceding the END statement. The ENDEF (End of Diskdef) macro generates the necessary uninitialized RAM areas which are located above your BIOS.

The form of the DISKDEF macro call is

```
DISKDEF  dn,fsc,lsc,[skf],bls,dks,dir,cks,ofs,[0]
```

where

```
dn      is the logical disk number, 0 to n-1
fsc     is the first physical sector number (0 or 1)
lsc     is the last sector number
skf     is the optional sector skew factor
bls     is the data allocation block size
dir     is the number of directory entries
cks     is the number of "checked" directory entries
ofs     is the track offset to logical track 00
[0]     is an optional 1.4 compatibility flag
```

The value "dn" is the drive number being defined with this DISKDEF macro invocation. The "fsc" parameter accounts for differing sector numbering systems, and is usually 0 or 1. The "lsc" is the last numbered sector on a track. When present, the "skf" parameter defines the sector skew factor which is used to create a sector translation table according to the skew. If the number of sectors is less than 256, a single-byte table is created, otherwise each translation table element occupies two bytes. No translation table is created if the skf parameter is omitted (or equal to 0). The "bls" parameter specifies the number of bytes allocated to each data block, and takes on the values 1024, 2048, 4096, 8192, or 16384. Generally, performance increases with larger data block sizes since there are fewer directory references and logically connected data records are physically close on the disk. Further, each directory entry addresses more data and the BIOS-resident ram space is reduced. The "dks" specifies the total disk size in "bls" units. That is, if the bls = 2048 and dks = 1000, then the total disk capacity is 2,048,000 bytes. If dks is greater than 255, then the block size parameter bls must be greater than 1024. The value of "dir" is the total number of directory entries which may exceed 255, if desired. The "cks" parameter determines the number of directory items to check on each directory scan, and is used internally to detect changed disks during system operation, where an intervening cold or warm start has not occurred (when this situation is detected, CP/M automatically marks the disk read/only so that data is not subsequently destroyed). Normally the value of cks = dir when the media is easily changed, as is the case with a floppy disk subsystem. If the disk is permanently mounted, then the value of cks is typically 0, since the probability of changing disks without a restart is quite low. The "ofs" value determines the number of tracks to skip when this particular drive is addressed, which can be used to reserve additional operating system space or to simulate several logical drives on a single large capacity physical drive. Finally, the [Ø] parameter is included when file compatibility is required with versions of 1.4 which have been modified for higher density disks. This parameter ensures that only 16K is allocated for each directory record, as was the case for previous versions. Normally, this parameter is not included.

For convenience and economy of table space, the special form

```
DISKDEF    i,j
```

gives disk i the same characteristics as a previously defined drive j. A standard four-drive single density system, which is compatible with version 1.4, is defined using the following macro invocations:

```
DISKS     4
 DISKDEF   Ø,1,26,6,1Ø24,243,64,64,2
 DISKDEF   1,Ø
 DISKDEF   2,Ø
 DISKDEF   3,Ø

 ....
 ENDEF
```

with all disks having the same parameter values of 26 sectors per track (numbered 1 through 26), with 6 sectors skipped between each access, 1Ø24 bytes per data block, 243 data blocks for a total of 243k byte disk capacity, 64 checked directory entries, and two operating system tracks.

The definitions given in the program shown above (lines 12 through 15) provide access to the largest disks addressable by CP/M 2.Ø. All disks have identical parameters, except that drives Ø and 2 skip three sectors on every data access, while disks 1 and 3 access each sector in sequence as the disk revolves (there may, however, be a transparent hardware skew factor on these drives).

The DISKS macro generates n "disk header blocks," starting at address DPBASE which is a label generated by the macro. Each disk header block contains sixteen bytes, and correspond, in sequence, to each of the defined drives. In the four drive standard system, for example, the DISKS macro generates a table of the form:

```
DPBASE  EQU  $
DPEØ:   DW   XLTØ,ØØØØH,ØØØØH,ØØØØH,DIRBUF,DPBØ,CSVØ,ALVØ
DPE1:   DW   XLTØ,ØØØØH,ØØØØH,ØØØØH,DIRBUF,DPBØ,CSV1,ALV1
DPE2:   DW   XLTØ,ØØØØH,ØØØØH,ØØØØH,DIRBUF,DPBØ,CSV2,ALV2
DPE3:   DW   XLTØ,ØØØØH,ØØØØH,ØØØØH,DIRBUF,DPBØ,CSV3,ALV3
```

where the DPE (disk parameter entry) labels are included for reference purposes to show the beginning table addresses for each drive Ø through 3. The values contained within the disk parameter header are described in detail in the CP/M 2.Ø Alteration Guide, but basically address the translation vector for the drive (all reference XLTØ, which is the translation vector for drive Ø in the above example), followed by three 16-bit "scratch" addresses, followed by the directory buffer address, disk parameter block address, check vector address, and allocation vector address. The check and allocation vector addresses are generated by the ENDEF macro in the ram area following the BIOS code and tables.

The SELDSK function is extended somewhat in version 2.0. In particular, the selected disk number is passed to the BIOS in register C, as before, and the SELDSK subroutine performs the appropriate software or hardware actions to select the disk. Version 2.0, however, also requires the SELDSK subroutine to return the address of the selected disk parameter header (DPE0, DPE1, DPE2, or DPE3, in the above example) in register HL. If SELDSK returns the value HL = 0000H, then the BDOS assumes the disk does not exist, and prints a select error mesage at the terminal. Program lines 22 through 36 give a sample CP/M 2.0 SELDSK subroutine, showing only the disk parameter header address calculation.

The subroutine SECTRAN is also included in version 2.0 which performs the actual logical to physical sector translation. In earlier versions of CP/M, the sector translation process was a part of the BDOS, and set to skip six sectors between each read. Due differing rotational speeds of various disks, the translation function has become a part of the BIOS in version 2.0. Thus, the BDOS sends sequential sector numbers to SECTRAN, starting at sector number 0. The SECTRAN subroutine uses the sequential sector number to produce a translated sector number which is returned to the BDOS. The BDOS subsequently sends the translated sector number to SELSEC before the actual read or write is performed. Note that many controllers have the capability to record the sector skew on the disk itself, and thus there is no translation necessary. In this case, the "skf" parameter is omitted in the macro call, and SECTRAN simply returns the same value which it receives. The table shown below, for example, is constructed when the standard skew factor skf = 6 is specified in the DISKDEF macro call:

```
XLT0:  DB    1,7,13,19,25,5,11,17,23,3,9,15,21
       DB    2,8,14,20,26,6,12,18,24,4,10,16,22
```

If SECTRAN is required to translate a sector, then the following process takes place. The sector to translate is received in register pair BC. Only the C register is significant if the sector value does not exceed 255 (B = 00 in this case). Register pair DE addresses the sector translate table for this drive, determined by a previous call on SELDSK, corresponding to the first element of a disk parameter header (XLT0 in the case shown above). The SECTRAN subroutine then fetches the translated sector number by adding the input sector number to the base of the translate table, to get the indexed translate table address (see lines 46, 47, and 48 in the above program). The value at this location is then returned in register L. Note that if the number of sectors exceeds 255, the translate table contains 16-bit elements whose value must be returned in HL.

Following the ENDEF macro call, a number of uninitialized data areas are defined. These data areas need not be a part of the BIOS which is loaded upon cold start, but must be available between the BIOS and the end of memory. The size of the uninitialized RAM area is determined by EQU statements generated by the ENDEF macro. For a standard four-drive system, the ENDEF macro might produce

```
4C72 =          BEGDAT EQU $
                (data areas)
4DB0 =          ENDDAT EQU $
013C =          DATSIZ EQU $-BEGDAT
```

which indicates that uninitialized RAM begins at location 4C72H, ends at 4DB0H-1, and occupies 013CH bytes. You must ensure that these addresses are free for use after the system is loaded.

CP/M 2.0 is also easily adapated to disk subsystems whose sector size is a multiple of 128 bytes. Information is provided by the BDOS on sector write operations which eliminates the need for pre-read operations, thus allowing blocking and deblocking to take place at the BIOS level.

See the "CP/M 2.0 Alteration Guide" for additional details concerning tailoring your CP/M system to your particular hardware.

DIGITAL RESEARCH®

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

CP/M 2.2 ALTERATION GUIDE

Copyright

Disclaimer

Trademarks

CP/M 2.2 ALTERATION GUIDE

1. INTRODUCTION

The standard CP/M system assumes operation on an Intel MDS-800 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 any IBM-3741 format compatible drive controller and other peripheral devices.

Although standard CP/M 2.0 is configured for single density floppy disks, field-alteration features allow adaptation to a wide variety of disk subsystems from single drive minidisks through high-capacity "hard disk" systems. In order to simplify the following adaptation process, we assume that CP/M 2.0 will first be configured for single density floppy disks where minimal editing and debugging tools are available. If an earlier version of CP/M is available, the customizing process is eased considerably. In this latter case, you may wish to briefly review the system generation process, and skip to later sections which discuss system alteration for non-standard disk systems.

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 your new BIOS into CP/M.

If CP/M is being tailored to your computer system for the first time, the new BIOS requires some relatively simple software development and testing. The standard BIOS is listed in Appendix B, and can be used as a model for the customized package. A skeletal version of the BIOS is given in Appendix C which can serve as the basis 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 can be 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 D. 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 E 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 shown with a following "H" which denotes the hexadecimal radix, and are given for a 20K 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 - 20K. Values for b in various standard memory sizes are

```
24K:     b = 24K - 20K =  4K = 1000H
32K:     b = 32K - 20K = 12K = 3000H
40K:     b = 40K - 20K = 20K = 5000H
48K:     b = 48K - 20K = 28K = 7000H
56K:     b = 56K - 20K = 36K = 9000H
62K:     b = 62K - 20K = 42K = A800H
64K:     b = 64K - 20K = 44K = B000H
```

Note: The standard distribution version of CP/M is set for operation within a 20K memory system. Therefore, you must first bring up the 20K 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 3380H. Code GETSYS so that it starts at location 100H (base of the TPA), as shown in the first part of Appendix d.

(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 3380H (the operating system actually starts 128 bytes later at 3400H).

(4) Review Section 4 and write the PUTSYS program which writes memory starting at 3380H 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 D.

(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 B, and write a simple version which performs a similar function for the customized environment. Use the program given in Appendix C 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 accidently 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 placed between locations 4A00H and 4FFFH. 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 3380H, 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 4A00H. The cold start routine will initialize page zero, then jump to the CCP at location 3400H which will call the BDOS, which will call the CBIOS. The CBIOS will be asked by the CCP to read sixteen sectors on track 2, and if successful, CP/M will type "A>", the system prompt.

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 retry.

(12) Then 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 Ø, 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 (ctl-C is sufficient) when the 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 procedures. You should take the time to become familiar 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), 1979
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 a additional disk drives, if desired. 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, (assuming media compatiblity, of course) 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 relocator) and place this memory image into a named disk file. The disk file can then be loaded, examined, patched, and replaced using the debugger, and system generation program. For further details on the operation of these programs, see the "Guide to CP/M Features and Facilities" manual.

Your CBIOS and BOOT can be modified using ED, 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.

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 2.0
READY FOR "SYSGEN" OR
"SAVE 34 CPMxx.COM"
```

At this point, an image of a CP/M in the TPA configured for the requested memory size. The memory image is at location 0900H through 227FH. (i.e., The BOOT is at 0900H, the CCP is at 980H, the BDOS starts at 1180H, and the BIOS is at 1F80H.) 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 34 CPMxx.COM
```

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
2300  0100
-                             (The DDT prompt)
```

You can then use the display and disassembly commands to examine

(All Information Contained Herein is Proprietary to Digital Research.)

portions of the memory image between 900H and 227FH. 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 20K system, for example, the CCP resides at the CP/M address 3400H, but is placed into memory at 980H by the SYSGEN program. Thus, the negative bias, denoted by n, satisfies

3400H + n = 980H, or n = 980H - 3400H

Assuming two's complement arithmetic, n = D580H, which can be checked by

3400H + D580H = 10980H = 0980H (ignoring high-order overflow).

Note that for larger systems, n satisfies

(3400H+b) + n = 980H, or
n = 980H - (3400H + b), or
n = D580H - b.

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

memory size	bias b	negative offset n
20K	0000H	D580H - 0000H = D580H
24K	1000H	D580H - 1000H = C580H
32K	3000H	D580H - 3000H = A580H
40K	5000H	D580H - 5000H = 8580H
48K	7000H	D580H - 7000H = 6580H
56K	9000H	D580H - 9000H = 4580H
62K	A800H	D580H - A800H = 2D80H
64K	B000H	D580H - B000H = 2580H

Assume, for example, that you want to locate the address x within the memory image loaded under DDT in a 20K 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

H3400,D580

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 the BIOS located at (4A00H+b)-n which, when you use the H command, produces an actual address of 1F80H. The disassembly command would thus be

L1F80

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 "n", then to calculate the bias (m) use the command:

H900,n	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-in the BOOT, give the command:

ICBOOT.HEX	Input file CBOOT.HEX

Then:

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

You may now examine your CBOOT with:

L900

We are now ready to replace the CBIOS. Examine the area at 1F80H where the original version of the CBIOS resides. Then type

ICBIOS.HEX	Ready the "hex" file for loading

assume that your CBIOS is being integrated into a 20K CP/M system, and thus is origined at location 4A00H. In order to properly locate the CBIOS in the memory image under DDT, we must apply the negative bias n for a 20K system when loading the hex file. This is accomplished by typing

RD580	Read the file with bias D580H

Upon completion of the read, re-examine the area where the CBIOS has been loaded (use an "L1F80" command), to ensure that is was loaded properly. When you are satisfied that the change has been made, return from DDT using a control-C or "G0" command.

Now use SYSGEN to replace 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 2.0            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
                              Place a scratch diskette in
                              drive B, then type return.
FUNCTION COMPLETE
DESTINATION DRIVE NAME (OR RETURN TO REBOOT)
```

Place the scratch diskette in your drive A, and then perform a coldstart to bring up the new CP/M system you have configured.

Test the new CP/M system, and place the Digital Research copyright notice on the diskette, as specified in your Licensing Agreement:

Copyright (c), 1979
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 3380H
;    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,3380H        ;SET STACK POINTER TO SCRATCH AREA
        LXI     H, 3380H        ;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 and listed in Appendix C for reference purposes, with an assumed origin of 100H. 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 D. The register pair HL become 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 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 the Appendix.

5. DISKETTE ORGANIZATION

The sector allocation for the standard distribution version of CP/M is given here for reference purposes. The first sector (see table on the following page) 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 BOOT, has the responsibility of bringing the remaining sectors into memory starting at location 3400H+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 3400H+b.

As an example, the Intel MDS-800 hardware cold start loader brings track 0, sector 1 into absolute address 3000H. Upon loading this sector, control transfers to location 3000H, where the bootstrap operation commences by loading the remainder of tracks 0, and all of track 1 into memory, starting at 3400H+b. The user should note that this bootstrap loader is of little use in a non-MDS environment, although it is useful to examine it since some of the boot actions will have to be duplicated in your cold start loader.

Track#	Sector#	Page#	Memory Address	CP/M Module name
00	01		(boot address)	Cold Start Loader
00	02	00	3400H+b	CCP
"	03	"	3480H+b	"
"	04	01	3500H+b	"
"	05	"	3580H+b	"
"	06	02	3600H+b	"
"	07	"	3680H+b	"
"	08	03	3700H+b	"
"	09	"	3780H+b	"
"	10	04	3800H+b	"
"	11	"	3880H+b	"
"	12	05	3900H+b	"
"	13	"	3980H+b	"
"	14	06	3A00H+b	"
"	15	"	3A80H+b	"
"	16	07	3B00H+b	"
00	17	"	3B80H+b	CCP
00	18	08	3C00H+b	BDOS
"	19	"	3C80H+b	"
"	20	09	3D00H+b	"
"	21	"	3D80H+b	"
"	22	10	3E00H+b	"
"	23	"	3E80H+b	"
"	24	11	3F00H+b	"
"	25	"	3F80H+b	"
"	26	12	4000H+b	"
01	01	"	4080H+b	"
"	02	13	4100H+b	"
"	03	"	4180H+b	"
"	04	14	4200H+b	"
"	05	"	4280H+b	"
"	06	15	4300H+b	"
"	07	"	4380H+b	"
"	08	16	4400H+b	"
"	09	"	4480H+b	"
"	10	17	4500H+b	"
"	11	"	4580H+b	"
"	12	18	4600H+b	"
"	13	"	4680H+b	"
"	14	19	4700H+b	"
"	15	"	4780H+b	"
"	16	20	4800H+b	"
"	17	"	4880H+b	"
"	18	21	4900H+b	"
01	19	"	4980H+b	BDOS
01	20	22	4A00H+b	BIOS
"	21	"	4A80H+b	"
"	23	23	4B00H+b	"
"	24	"	4B80H+b	"
"	25	24	4C00H+b	"
01	26	"	4C80H+b	BIOS
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" located at 4A00H+b, as shown below (see Appendices B and C, as well). The jump vector is a sequence of 17 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.

The jump vector at 4A00H+b takes the form shown below, where the individual jump addresses are given to the left:

```
4A00H+b     JMP BOOT       ; ARRIVE HERE FROM COLD START LOAD
4A03H+b     JMP WBOOT      ; ARRIVE HERE FOR WARM START
4A06H+b     JMP CONST      ; CHECK FOR CONSOLE CHAR READY
4A09H+b     JMP CONIN      ; READ CONSOLE CHARACTER IN
4A0CH+b     JMP CONOUT     ; WRITE CONSOLE CHARACTER OUT
4A0FH+b     JMP LIST       ; WRITE LISTING CHARACTER OUT
4A12H+b     JMP PUNCH      ; WRITE CHARACTER TO PUNCH DEVICE
4A15H+b     JMP READER     ; READ READER DEVICE
4A18H+b     JMP HOME       ; MOVE TO TRACK 00 ON SELECTED DISK
4A1BH+b     JMP SELDSK     ; SELECT DISK DRIVE
4A1EH+b     JMP SETTRK     ; SET TRACK NUMBER
4A21H+b     JMP SETSEC     ; SET SECTOR NUMBER
4A24H+b     JMP SETDMA     ; SET DMA ADDRESS
4A27H+b     JMP READ       ; READ SELECTED SECTOR
4A2AH+b     JMP WRITE      ; WRITE SELECTED SECTOR
4A2DH+b     JMP LISTST     ; RETURN LIST STATUS
4A30H+b     JMP SECTRAN    ; SECTOR TRANSLATE SUBROUTINE
```

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: the system (re)initialization which results from calls on BOOT and WBOOT, simple character I/O performed by calls on CONST, CONIN, CONOUT, LIST, PUNCH, READER, and LISTST, and diskette I/O performed by calls on HOME, SELDSK, SETTRK, SETSEC, SETDMA, READ, WRITE, and SECTRAN.

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 for an input device 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 may be used by PIP, but not the BDOS). Further, the LISTST entry is used currently only by DESPOOL, and 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 may 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 just simply return, and the READER routine can return with a 1AH (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 (RDR:)
- 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 (PUN:)
- 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 which 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 retries (10 is standard) 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 responsibilites 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 3400H+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 4A03H+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 3C06H+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 3400H+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 and 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 parity bit (high order bit) to zero. 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, and so-forth up to 15 for drive P (the standard CP/M distribution version supports four drives). On each disk select, SELDSK must return in HL the base address of a 16-byte area, called the Disk Parameter Header, described in the Section 10. For standard floppy disk drives, the contents of the header and associated tables does not change, and thus the program segment included in the sample CBIOS performs this operation automatically. If there is an attempt to select a non-existent drive, SELDSK returns HL=0000H as an error indicator. Although SELDSK must return the header address on each call, it is advisable to postpone the actual physical disk select operation until an I/O function (seek, read or write) is actually performed, since disk selects often occur without utimately 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 BC 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 BC can take on values in the range 0-76 corresponding to valid track numbers for standard floppy disk drives, and 0-65535 for non-standard disk subsystems.

SETSEC — Register BC 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 — Register BC contains the DMA (disk 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 retries 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 ctl-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.

LISTST Return the ready status of the list device. Used by the DESPOOL program to improve console response during its operation. The value 00 is returned in A if the list device is not ready to accept a character, and 0FFH if a character can be sent to the printer. Note that a 00 value always suffices.

SECTRAN Performs sector logical to physical sector translation in order to improve the overall response of CP/M. Standard CP/M systems are shipped with a "skew factor" of 6, where six physical sectors are skipped between each logical read operation. This skew factor allows enough time between sectors for most programs to load their buffers without missing the next sector. In particular computer systems which use fast processors, memory, and disk subsystems, the skew factor may be changed to improve overall response. Note, however, that you should maintain a single density IBM compatible version of CP/M for information transfer into and out of your computer system, using a skew factor of 6. In general, SECTRAN receives a logical sector number in BC, and a translate table address in DE. The sector number is used as an index into the translate table, with the resulting physical sector number in HL. For standard systems, the tables and indexing code is provided in the CBIOS and need not be changed.

7. A SAMPLE BIOS

The program shown in Appendix C 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 WAITIO subroutines. 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 D 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, between locations ØØH and ØFFH, 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
ØØØØH - ØØØ2H	Contains a jump instruction to the warm start entry point at location 4AØ3H+b. This allows a simple programmed restart (JMP ØØØØH) or manual restart from the front panel.
ØØØ3H - ØØØ3H	Contains the Intel standard IOBYTE, which is optionally included in the user's CBIOS, as described in Section 6.
ØØØ4H - ØØØ4H	Current default drive number (Ø=A,...,15=P).
ØØØ5H - ØØØ7H	Contains a jump instruction to the BDOS,and serves two purposes: JMP ØØØ5H provides the primary entry point to the BDOS, as described in the manual "CP/M Interface Guide," and LHLD ØØØ6H 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.
ØØØ8H - ØØ27H	(interrupt locations 1 through 5 not used)
ØØ3ØH - ØØ37H	(interrupt location 6, not currently used - reserved)
ØØ38H - ØØ3AH	Restart 7 - Contains a jump instruction into the DDT or SID program when running in debug mode for programmed breakpoints, but is not otherwise used by CP/M.
ØØ3BH - ØØ3FH	(not currently used - reserved)
ØØ4ØH - ØØ4FH	16 byte area reserved for scratch by CBIOS, but is not used for any purpose in the distribution version of CP/M
ØØ5ØH - ØØ5BH	(not currently used - reserved)
ØØ5CH - ØØ7CH	default file control block produced for a transient program by the Console Command Processor.
ØØ7DH - ØØ7FH	Optional default random record position

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 set-up 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. DISK PARAMETER TABLES.

Tables are included in the BIOS which describe the particular characteristics of the disk subsystem used with CP/M. These tables can be either hand-coded, as shown in the sample CBIOS in Appendix C, or automatically generated using the DISKDEF macro library, as shown in Appendix B. The purpose here is to describe the elements of these tables.

In general, each disk drive has an associated (16-byte) disk parameter header which both contains information about the disk drive and provides a scratchpad area for certain BDOS operations. The format of the disk parameter header for each drive is shown below

Disk Parameter Header

XLT	0000	0000	0000	DIRBUF	DPB	CSV	ALV
16b	16b	16b	16b	16b	16b	16b	16b

where each element is a word (16-bit) value. The meaning of each Disk Parameter Header (DPH) element is

XLT — Address of the logical to physical translation vector, if used for this particular drive, or the value 0000H if no sector translation takes place (i.e, the physical and logical sector numbers are the same). Disk drives with identical sector skew factors share the same translate tables.

0000 — Scratchpad values for use within the BDOS (initial value is unimportant).

DIRBUF — Address of a 128 byte scratchpad area for directory operations within BDOS. All DPH's address the same scratchpad area.

DPB — Address of a disk parameter block for this drive. Drives with identical disk characteristics address the same disk parameter block.

CSV — Address of a scratchpad area used for software check for changed disks. This address is different for each DPH.

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

Given n disk drives, the DPH's are arranged in a table whose first row of 16 bytes corresponds to drive 0, with the last row corresponding to drive n-1. The table thus appears as

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

where the label DPBASE defines the base address of the DPH table.

A responsibility of the SELDSK subroutine is to return the base address of the DPH for the selected drive. The following sequence of operations returns the table address, with a 0000H returned if the selected drive does not exist.

```
NDISKS   EQU     4   ;NUMBER OF DISK DRIVES
......
SELDSK:
         ;SELECT DISK GIVEN BY BC
         LXI     H,0000H  ;ERROR CODE
         MOV     A,C      ;DRIVE OK?
         CPI     NDISKS   ;CY IF SO
         RNC              ;RET IF ERROR
         ;NO ERROR, CONTINUE
         MOV     L,C      ;LOW(DISK)
         MOV     H,B      ;HIGH(DISK)
         DAD     H        ;*2
         DAD     H        ;*4
         DAD     H        ;*8
         DAD     H        ;*16
         LXI     D,DPBASE ;FIRST DPH
         DAD     D        ;DPH(DISK)
         RET
```

The translation vectors (XLT 00 through XLTn-1) are located elsewhere in the BIOS, and simply correspond one-for-one with the logical sector numbers zero through the sector count-1. The Disk Parameter Block (DPB) for each drive is more complex. A particular DPB, which is addressed by one or more DPH's, takes the general form

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

where each is a byte or word value, as shown by the "8b" or "16b" indicator below the field.

SPT is the total number of sectors per track

BSH is the data allocation block shift factor, determined by the data block allocation size.

EXM	is the extent mask, determined by the data block allocation size and the number of disk blocks.
DSM	determines the total storage capacity of the disk drive
DRM	determines the total number of directory entries which can be stored on this drive ALØ,AL1 determine reserved directory blocks.
CKS	is the size of the directory check vector
OFF	is the number of reserved tracks at the beginning of the (logical) disk.

The values of BSH and BLM determine (implicitly) the data allocation size BLS, which is not an entry in the disk parameter block. Given that the designer has selected a value for BLS, the values of BSH and BLM are shown in the table below

BLS	BSH	BLM
1,Ø24	3	7
2,Ø48	4	15
4,Ø96	5	31
8,192	6	63
16,384	7	127

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

BLS	DSM < 256	DSM > 255
1,Ø24	Ø	N/A
2,Ø48	1	Ø
4,Ø96	3	1
8,192	7	3
16,384	15	7

The value of DSM is the maximum data block number supported by this particular drive, measured in BLS units. The product BLS times (DSM+1) is the total number of bytes held by the drive and, of course, must be within the capacity of the physical disk, not counting the reserved operating system tracks.

The DRM entry is the one less than the total number of directory entries, which can take on a 16-bit value. The values of ALØ and AL1, however, are determined by DRM. The two values ALØ and AL1 can together be considered a string of 16-bits, as shown below.

```
---------------------------------------------------
|          AL0            |           AL1         |
---------------------------------------------------
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
---------------------------------------------------
 00 01 02 03 04 05 06 07 08 09 10 11 12 13 14 15
```

where position 00 corresponds to the high order bit of the byte labelled AL0, and 15 corresponds to the low order bit of the byte labelled AL1. Each bit position reserves a data block for number of directory entries, thus allowing a total of 16 data blocks to be assigned for directory entries (bits are assigned starting at 00 and filled to the right until position 15). Each directory entry occupies 32 bytes, resulting in the following table

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

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

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

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

To complete the discussion of the DPB, recall that several DPH's can address the same DPB if their drive characteristics are identical. Further, the DPB can be dynamically changed when a new drive is addressed by simply changing the pointer in the DPH since the BDOS copies the DPB values to a local area whenever the SELDSK function is invoked.

Returning back to the DPH for a particular drive, note that the two address values CSV and ALV remain. Both addresses reference an area of uninitialized memory following the BIOS. The areas must be unique for each drive, and the size of each area is determined by the values in the DPB.

The size of the area addressed by CSV is CKS bytes, which is sufficient to hold the directory check information for this particular drive. If CKS = (DRM+1)/4, then you must reserve (DRM+1)/4 bytes for directory check use. If CKS = 0, then no storage is reserved.

The size of the area addressed by ALV is determined by the maximum number of data blocks allowed for this particular disk, and is computed as (DSM/8)+1.

The CBIOS shown in Appendix C demonstrates an instance of these tables for standard 8" single density drives. It may be useful to examine this program, and compare the tabular values with the definitions given above.

11. THE DISKDEF MACRO LIBRARY.

A macro library is shown in Appendix F, called DISKDEF, which greatly simplifies the table construction process. You must have access to the MAC macro assembler, of course, to use the DISKDEF facility, while the macro library is included with all CP/M 2.0 distribution disks.

A BIOS disk definition consists of the following sequence of macro statements:

```
MACLIB   DISKDEF
......
DISKS    n
DISKDEF  0,...
DISKDEF  1,...
......
DISKDEF  n-1
......
ENDEF
```

where the MACLIB statement loads the DISKDEF.LIB file (on the same disk as your BIOS) into MAC's internal tables. The DISKS macro call follows, which specifies the number of drives to be configured with your system, where n is an integer in the range 1 to 16. A series of DISKDEF macro calls then follow which define the characteristics of each logical disk, 0 through n-1 (corresponding to logical drives A through P). Note that the DISKS and DISKDEF macros generate the in-line fixed data tables described in the previous section, and thus must be placed in a non-executable portion of your BIOS, typically directly following the BIOS jump vector.

The remaining portion of your BIOS is defined following the DISKDEF macros, with the ENDEF macro call immediately preceding the END statement. The ENDEF (End of Diskdef) macro generates the necessary uninitialized RAM areas which are located in memory above your BIOS.

The form of the DISKDEF macro call is

```
DISKDEF  dn,fsc,lsc,[skf],bls,dks,dir,cks,ofs,[0]
```

where

```
dn      is the logical disk number, 0 to n-1
fsc     is the first physical sector number (0 or 1)
lsc     is the last sector number
skf     is the optional sector skew factor
bls     is the data allocation block size
dir     is the number of directory entries
cks     is the number of "checked" directory entries
ofs     is the track offset to logical track 00
[0]     is an optional 1.4 compatibility flag
```

The value "dn" is the drive number being defined with this DISKDEF macro invocation. The "fsc" parameter accounts for differing sector numbering systems, and is usually 0 or 1. The "lsc" is the last numbered sector on a track. When present, the "skf" parameter defines the sector skew factor which is used to create a sector translation table according to the skew. If the number of sectors is less than 256, a single-byte table is created, otherwise each translation table element occupies two bytes. No translation table is created if the skf parameter is omitted (or equal to 0). The "bls" parameter specifies the number of bytes allocated to each data block, and takes on the values 1024, 2048, 4096, 8192, or 16384. Generally, performance increases with larger data block sizes since there are fewer directory references and logically connected data records are physically close on the disk. Further, each directory entry addresses more data and the BIOS-resident ram space is reduced. The "dks" specifies the total disk size in "bls" units. That is, if the bls = 2048 and dks = 1000, then the total disk capacity is 2,048,000 bytes. If dks is greater than 255, then the block size parameter bls must be greater than 1024. The value of "dir" is the total number of directory entries which may exceed 255, if desired. The "cks" parameter determines the number of directory items to check on each directory scan, and is used internally to detect changed disks during system operation, where an intervening cold or warm start has not occurred (when this situation is detected, CP/M automatically marks the disk read/only so that data is not subsequently destroyed). As stated in the previous section, the value of cks = dir when the media is easily changed, as is the case with a floppy disk subsystem. If the disk is permanently mounted, then the value of cks is typically 0, since the probability of changing disks without a restart is quite low. The "ofs" value determines the number of tracks to skip when this particular drive is addressed, which can be used to reserve additional operating system space or to simulate several logical drives on a single large capacity physical drive. Finally, the [0] parameter is included when file compatibility is required with versions of 1.4 which have been modified for higher density disks. This parameter ensures that only 16K is allocated for each directory record, as was the case for previous versions. Normally, this parameter is not included.

For convenience and economy of table space, the special form

```
DISKDEF    i,j
```

gives disk i the same characteristics as a previously defined drive j. A standard four-drive single density system, which is compatible with version 1.4, is defined using the following macro invocations:

```
DISKS       4
DISKDEF     0,1,26,6,1024,243,64,64,2
DISKDEF     1,0
DISKDEF     2,0
DISKDEF     3,0
....
ENDEF
```

with all disks having the same parameter values of 26 sectors per track (numbered 1 through 26), with 6 sectors skipped between each access, 1024 bytes per data block, 243 data blocks for a total of 243k byte disk capacity, 64 checked directory entries, and two operating system tracks.

The DISKS macro generates n Disk Parameter Headers (DPH's), starting at the DPH table address DPBASE generated by the macro. Each disk header block contains sixteen bytes, as described above, and correspond one-for-one to each of the defined drives. In the four drive standard system, for example, the DISKS macro generates a table of the form:

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

where the DPH labels are included for reference purposes to show the beginning table addresses for each drive 0 through 3. The values contained within the disk parameter header are described in detail in the previous section. The check and allocation vector addresses are generated by the ENDEF macro in the ram area following the BIOS code and tables.

Note that if the "skf" (skew factor) parameter is omitted (or equal to 0), the translation table is omitted, and a 0000H value is inserted in the XLT position of the disk parameter header for the disk. In a subsequent call to perform the logical to physical translation, SECTRAN receives a translation table address of DE = 0000H, and simply returns the original logical sector from BC in the HL register pair. A translate table is constructed when the skf parameter is present, and the (non-zero) table address is placed into the corresponding DPH's. The table shown below, for example, is constructed when the standard skew factor skf = 6 is specified in the DISKDEF macro call:

```
XLT0:  DB    1,7,13,19,25,5,11,17,23,3,9,15,21
       DB    2,8,14,20,26,6,12,18,24,4,10,16,22
```

Following the ENDEF macro call, a number of uninitialized data areas are defined. These data areas need not be a part of the BIOS which is loaded upon cold start, but must be available between the BIOS and the end of memory. The size of the uninitialized RAM area is determined by EQU statements generated by the ENDEF macro. For a standard four-drive system, the ENDEF macro might produce

```
4C72 =          BEGDAT EQU $
                (data areas)
4DB0 =          ENDDAT EQU $
013C =          DATSIZ EQU $-BEGDAT
```

which indicates that uninitialized RAM begins at location 4C72H, ends at 4DB0H-1, and occupies 013CH bytes. You must ensure that these addresses are free for use after the system is loaded.

After modification, you can use the STAT program to check your drive characteristics, since STAT uses the disk parameter block to decode the drive information. The STAT command form

STAT d:DSK:

decodes the disk parameter block for drive d (d=A,...,P) and displays the values shown below:

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

Three examples of DISKDEF macro invocations are shown below with corresponding STAT parameter values (the last produces a full 8-megabyte system).

```
DISKDEF 0,1,58,,2048,256,128,128,2
r=4096, k=512, d=128, c=128, e=256, b=16, s=58, t=2

DISKDEF 0,1,58,,2048,1024,300,0,2
r=16384, k=2048, d=300, c=0, e=128, b=16, s=58, t=2

DISKDEF 0,1,58,,16384,512,128,128,2
r=65536, k=8192, d=128, c=128, e=1024, b=128, s=58, t=2
```

12. SECTOR BLOCKING AND DEBLOCKING.

Upon each call to the BIOS WRITE entry point, the CP/M BDOS includes information which allows effective sector blocking and deblocking where the host disk subsystem has a sector size which is a multiple of the basic 128-byte unit. The purpose here is to present a general-purpose algorithm which can be included within your BIOS which uses the BDOS information to perform the operations automatically.

Upon each call to WRITE, the BDOS provides the following information in register C:

```
0   =   normal sector write
1   =   write to directory sector
2   =   write to the first sector
        of a new data block
```

Condition 0 occurs whenever the next write operation is into a previously written area, such as a random mode record update, when the write is to other than the first sector of an unallocated block, or when the write is not into the directory area. Condition 1 occurs when a write into the directory area is performed. Condition 2 occurs when the first record (only) of a newly allocated data block is written. In most cases, application programs read or write multiple 128 byte sectors in sequence, and thus there is little overhead involved in either operation when blocking and deblocking records since pre-read operations can be avoided when writing records.

Appendix G lists the blocking and deblocking algorithms in skeletal form (this file is included on your CP/M disk). Generally, the algorithms map all CP/M sector read operations onto the host disk through an intermediate buffer which is the size of the host disk sector. Throughout the program, values and variables which relate to the CP/M sector involved in a seek operation are prefixed by "sek," while those related to the host disk system are prefixed by "hst." The equate statements beginning on line 29 of Appendix G define the mapping between CP/M and the host system, and must be changed if other than the sample host system is involved.

The entry points BOOT and WBOOT must contain the initialization code starting on line 57, while the SELDSK entry point must be augmented by the code starting on line 65. Note that although the SELDSK entry point computes and returns the Disk Parameter Header address, it does not physically selected the host disk at this point (it is selected later at READHST or WRITEHST). Further, SETTRK, SETTRK, and SETDMA simply store the values, but do not take any other action at this point. SECTRAN performs a trivial trivial function of returning the physical sector number.

The principal entry points are READ and WRITE, starting on lines 110 and 125, respectively. These subroutines take the place of your previous READ and WRITE operations.

The actual physical read or write takes place at either WRITEHST or READHST, where all values have been prepared: hstdsk is the host disk number, hsttrk is the host track number, and hstsec is the host sector number (which may require translation to a physical sector number). You must insert code at this point which performs the full host sector read or write into, or out of, the buffer at hstbuf of length hstsiz. All other mapping functions are performed by the algorithms.

This particular algorithm was tested using an 80 megabyte hard disk unit which was originally configured for 128 byte sectors, producing approximately 35 megabytes of formatted storage. When configured for 512 byte host sectors, usable storage increased to 57 megabytes, with a corresponding 400% improvement in overall response. In this situation, there is no apparent overhead involved in deblocking sectors, with the advantage that user programs still maintain the (less memory consuming) 128-byte sectors. This is primarily due, of course, to the information provided by the BDOS which eliminates the necessity for pre-read operations to take place.

APPENDIX A: THE MDS COLD START LOADER

```
                ;       MDS-800 Cold Start Loader for CP/M 2.0
                ;
                ;       Version 2.0 August, 1979
                ;
0000 =          false   equ     0
ffff =          true    equ     not false
0000 =          testing equ     false
                ;
                        if      testing
                bias    equ     03400h
                        endif
                        if      not testing
0000 =          bias    equ     0000h
                        endif
0000 =          cpmb    equ     bias            ;base of dos load
0806 =          bdos    equ     806h+bias       ;entry to dos for calls
1880 =          bdose   equ     1880h+bias      ;end of dos load
1600 =          boot    equ     1600h+bias      ;cold start entry point
1603 =          rboot   equ     boot+3          ;warm start entry point
                ;
3000                    org     3000h   ;loaded here by hardware
                ;
1880 =          bdosl   equ     bdose-cpmb
0002 =          ntrks   equ     2               ;tracks to read
0031 =          bdoss   equ     bdosl/128       ;# sectors in bdos
0019 =          bdos0   equ     25              ;# on track 0
0018 =          bdosl   equ     bdoss-bdos0     ;# 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
00ff =          bsw     equ     0ffh    ;boot switch
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:
3000 310001             lxi     sp,stack;in case of call to mon80
                ;       clear disk status
3003 db79               in      rtype
3005 db7b               in      rbyte
                ;       check if boot switch is off
                coldstart:
3007 dbff               in      bsw
3009 e602               ani     02h     ;switch on?
300b c20730             jnz     coldstart
```

```
              ;         clear the controller
3ØØe d37f               out     reset   ;logic cleared
              ;
              ;
3Ø1Ø Ø6Ø2               mvi     b,ntrks ;number of tracks to read
3Ø12 21423Ø             lxi     h,iopbØ
              ;
              start:
              ;
              ;         read first/next track into cpmb
3Ø15 7d                 mov     a,l
3Ø16 d379               out     ilow
3Ø18 7c                 mov     a,h
3Ø19 d37a               out     ihigh
3Ø1b db78     waitØ:    in      dstat
3Ø1d e6Ø4               ani     4
3Ø1f ca1b3Ø             jz      waitØ
              ;
              ;         check disk status
3Ø22 db79               in      rtype
3Ø24 e6Ø3               ani     11b
3Ø26 feØ2               cpi     2
              ;
                        if      testing
                        cnc     rmon8Ø  ;go to monitor if 11 or 1Ø
                        endif
                        if      not testing
3Ø28 d2ØØ3Ø             jnc     rstart  ;retry the load
                        endif
              ;
3Ø2b db7b               in      rbyte   ;i/o complete, check status
              ;         if not ready, then go to mon8Ø
3Ø2d 17                 ral
3Ø2e dcØfff             cc      rmon8Ø  ;not ready bit set
3Ø31 1f                 rar             ;restore
3Ø32 e61e               ani     1111Øb  ;overrun/addr err/seek/crc
              ;
                        if      testing
                        cnz     rmon8Ø  ;go to monitor
                        endif
                        if      not testing
3Ø34 c2ØØ3Ø             jnz     rstart  ;retry the load
                        endif
              ;
              ;
3Ø37 11Ø7ØØ             lxi     d,iopbl ;length of iopb
3Ø3a 19                 dad     d       ;addressing next iopb
3Ø3b Ø5                 dcr     b       ;count down tracks
3Ø3c c2153Ø             jnz     start
              ;
              ;
              ;         jmp boot, print message, set-up jmps
3Ø3f c3ØØ16             jmp     boot
              ;
              ;         parameter blocks
```

```
3Ø42 8Ø         iopbØ:  db      8Øh      ;iocw, no update
3Ø43 Ø4                 db      readf    ;read function
3Ø44 19                 db      bdosØ    ;# sectors to read trk Ø
3Ø45 ØØ                 db      Ø        ;track Ø
3Ø46 Ø2                 db      2        ;start with sector 2, trk Ø
3Ø47 ØØØØ               dw      cpmb     ;start at base of bdos
ØØØ7 =          iopbl   equ     $-iopbØ
                ;
3Ø49 8Ø         iopbl:  db      8Øh
3Ø4a Ø4                 db      readf
3Ø4b 18                 db      bdosl    ;sectors to read on track 1
3Ø4c Ø1                 db      1        ;track 1
3Ø4d Ø1                 db      1        ;sector 1
3Ø4e 8ØØc               dw      cpmb+bdosØ*128  ;base of second rd
3Ø5Ø                    end
```

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

```
                ;       mds-800 i/o drivers for cp/m 2.0
                ;       (four drive single density version)
                ;
                ;       version 2.0 august, 1979
                ;
0014 =          vers    equ     20      ;version 2.0
                ;
                ;       copyright (c) 1979
                ;       digital research
                ;       box 579, pacific grove
                ;       california, 93950
                ;
4a00                    org     4a00h   ;base of bios in 20k system
3400 =          cpmb    equ     3400h   ;base of cpm ccp
3c06 =          bdos    equ     3c06h   ;base of bdos in 20k system
1600 =          cpml    equ     $-cpmb  ;length (in bytes) of cpm system
002c =          nsects  equ     cpml/128;number of sectors to load
0002 =          offset  equ     2       ;number of disk tracks used by cp
0004 =          cdisk   equ     0004h   ;address of last logged disk
0080 =          buff    equ     0080h   ;default buffer address
000a =          retry   equ     10      ;max retries on disk i/o before e
                ;
                ;       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 bloc
                ;       mds, which is used to perform subsequent reads an
                ;       seldsk  select disk given by reg-c (0,1,2...)
                ;       settrk  set track address (0,...76) for sub r/w
                ;       setsec  set sector address (1,...,26)
                ;       setdma  set subsequent dma address (initially 80h
                ;
                ;       read/write assume previous calls to set i/o parms
                ;       read    read track/sector to preset dma address
                ;       write   write track/sector from preset dma addres
                ;
                ;       jump vector for indiviual routines
 4a00 c3b34a            jmp     boot
 4a03 c3c34a    wboote: jmp     wboot
 4a06 c3614b            jmp     const
 4a09 c3644b            jmp     conin
 4a0c c36a4b            jmp     conout
```

```
4a0f c36d4b            jmp     list
4a12 c3724b            jmp     punch
4a15 c3754b            jmp     reader
4a18 c3784b            jmp     home
4a1b c37d4b            jmp     seldsk
4a1e c3a74b            jmp     settrk
4a21 c3ac4b            jmp     setsec
4a24 c3bb4b            jmp     setdma
4a27 c3c14b            jmp     read
4a2a c3ca4b            jmp     write
4a2d c3704b            jmp     listst  ;list status
4a30 c3b14b            jmp     sectran
                ;
                       maclib  diskdef ;load the disk definition library
                       disks   4       ;four disks
4a33+=          dpbase equ     $       ;base of disk parameter blocks
4a33+824a00     dpe0:  dw      xlt0,0000h      ;translate table
4a37+000000            dw      0000h,0000h     ;scratch area
4a3b+6e4c73            dw      dirbuf,dpb0     ;dir buff,parm block
4a3f+0d4dee            dw      csv0,alv0       ;check, alloc vectors
4a43+824a00     dpe1:  dw      xlt1,0000h      ;translate table
4a47+000000            dw      0000h,0000h     ;scratch area
4a4b+6e4c73            dw      dirbuf,dpb1     ;dir buff,parm block
4a4f+3c4d1d            dw      csv1,alv1       ;check, alloc vectors
4a53+824a00     dpe2:  dw      xlt2,0000h      ;translate table
4a57+000000            dw      0000h,0000h     ;scratch area
4a5b+6e4c73            dw      dirbuf,dpb2     ;dir buff,parm block
4a5f+6b4d4c            dw      csv2,alv2       ;check, alloc vectors
4a63+824a00     dpe3:  dw      xlt3,0000h      ;translate table
4a67+000000            dw      0000h,0000h     ;scratch area
4a6b+6e4c73            dw      dirbuf,dpb3     ;dir buff,parm block
4a6f+9a4d7b            dw      csv3,alv3       ;check, alloc vectors
                       diskdef 0,1,26,6,1024,243,64,64,offset
4a73+=          dpb0   equ     $               ;disk parm block
4a73+1a00              dw      26              ;sec per track
4a75+03                db      3               ;block shift
4a76+07                db      7               ;block mask
4a77+00                db      0               ;extnt mask
4a78+f200              dw      242             ;disk size-1
4a7a+3f00              dw      63              ;directory max
4a7c+c0                db      192             ;alloc0
4a7d+00                db      0               ;alloc1
4a7e+1000              dw      16              ;check size
4a80+0200              dw      2               ;offset
4a82+=          xlt0   equ     $               ;translate table
4a82+01                db      1
4a83+07                db      7
4a84+0d                db      13
4a85+13                db      19
4a86+19                db      25
4a87+05                db      5
4a88+0b                db      11
4a89+11                db      17
4a8a+17                db      23
4a8b+03                db      3
```

```
4a8c+09                 db      9
4a8d+0f                 db      15
4a8e+15                 db      21
4a8f+02                 db      2
4a90+08                 db      8
4a91+0e                 db      14
4a92+14                 db      20
4a93+1a                 db      26
4a94+06                 db      6
4a95+0c                 db      12
4a96+12                 db      18
4a97+18                 db      24
4a98+04                 db      4
4a99+0a                 db      10
4a9a+10                 db      16
4a9b+16                 db      22
                        diskdef 1,0
4a73+=          dpb1    equ     dpb0    ;equivalent parameters
001f+=          als1    equ     als0    ;same allocation vector size
0010+=          css1    equ     css0    ;same checksum vector size
4a82+=          xlt1    equ     xlt0    ;same translate table
                        diskdef 2,0
4a73+=          dpb2    equ     dpb0    ;equivalent parameters
001f+=          als2    equ     als0    ;same allocation vector size
0010+=          css2    equ     css0    ;same checksum vector size
4a82+=          xlt2    equ     xlt0    ;same translate table
                        diskdef 3,0
4a73+=          dpb3    equ     dpb0    ;equivalent parameters
001f+=          als3    equ     als0    ;same allocation vector size
0010+=          css3    equ     css0    ;same checksum vector size
4a82+=          xlt3    equ     xlt0    ;same translate table
                ;       endef occurs at end of assembly
                ;
                ;       end of controller - independent code, the remaini
                ;       are tailored to the particular operating environm
                ;       be altered for any system which differs from the
                ;
                ;       the following code assumes the mds monitor exists
                ;       and uses the i/o subroutines within the monitor
                ;
                ;       we also assume the mds system has four disk drive
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
                ;
                ;       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 o
f80c =          po      equ     0f80ch  ;punch char from c to punch devic
f80f =          lo      equ     0f80fh  ;list from c to list device
f812 =          csts    equ     0f812h  ;console status 00/ff to register
```

```
                ;
                ;         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 =          writf     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
4a9c 0d0a0a               db      cr,lf,lf
4a9f 3230                 db      '20'      ;sample memory size
4aa1 6b2043f              db      'k cp/m vers '
4aad 322e30               db      vers/10+'0','.',vers mod 10+'0'
4ab0 0d0a00               db      cr,lf,0
                ;
                boot:     ;print signon message and go to ccp
                ;         (note: mds boot initialized iobyte at 0003h)
4ab3 310001               lxi     sp,buff+80h
4ab6 219c4a               lxi     h,signon
4ab9 cdd34b               call    prmsg     ;print message
4abc af                   xra     a         ;clear accumulator
4abd 320400               sta     cdisk     ;set initially to disk a
4ac0 c30f4b               jmp     gocpm     ;go to cp/m
                ;
                ;
                wboot:; loader on track 0, sector 1, which will be skippe
                ;         read cp/m from disk - assuming there is a 128 byt
                ;         start.
                ;
4ac3 318000               lxi     sp,buff ;using dma - thus 80 thru ff ok f
                ;
4ac6 0e0a                 mvi     c,retry ;max retries
4ac8 c5                   push    b
                wboot0: ;enter here on error retries
4ac9 010034               lxi     b,cpmb    ;set dma address to start of disk
4acc cdbb4b               call    setdma
4acf 0e00                 mvi     c,0       ;boot from drive 0
4ad1 cd7d4b               call    seldsk
4ad4 0e00                 mvi     c,0
4ad6 cda74b               call    settrk    ;start with track 0
4ad9 0e02                 mvi     c,2       ;start reading sector 2
4adb cdac4b               call    setsec
                ;
                ;         read sectors, count nsects to zero
4ade c1                   pop     b         ;10-error count
4adf 062c                 mvi     b,nsects
```

```
                rdsec:  ;read next sector
4ae1 c5                 push    b       ;save sector count
4ae2 cdc14b             call    read
4ae5 c2494b             jnz     booterr ;retry if errors occur
4ae8 2a6c4c             lhld    iod     ;increment dma address
4aeb 118000             lxi     d,128   ;sector size
4aee 19                 dad     d       ;incremented dma address in hl
4aef 44                 mov     b,h
4af0 4d                 mov     c,l     ;ready for call to set dma
4af1 cdbb4b             call    setdma
4af4 3a6b4c             lda     ios     ;sector number just read
4af7 fe1a               cpi     26      ;read last sector?
4af9 da054b             jc      rd1
                ;       must be sector 26, zero and go to next track
4afc 3a6a4c             lda     iot     ;get track to register a
4aff 3c                 inr     a
4b00 4f                 mov     c,a     ;ready for call
4b01 cda74b             call    settrk
4b04 af                 xra     a       ;clear sector number
4b05 3c         rd1:    inr     a       ;to next sector
4b06 4f                 mov     c,a     ;ready for call
4b07 cdac4b             call    setsec
4b0a c1                 pop     b       ;recall sector count
4b0b 05                 dcr     b       ;done?
4b0c c2e14a             jnz     rdsec
                ;
                ;       done with the load, reset default buffer address
                gocpm:  ;(enter here from cold start boot)
                ;       enable rst0 and rst7
4b0f f3                 di
4b10 3e12               mvi     a,12h   ;initialize command
4b12 d3fd               out     revrt
4b14 af                 xra     a
4b15 d3fc               out     intc    ;cleared
4b17 3e7e               mvi     a,inte  ;rst0 and rst7 bits on
4b19 d3fc               out     intc
4b1b af                 xra     a
4b1c d3f3               out     icon    ;interrupt control
                ;
                ;       set default buffer address to 80h
4b1e 018000             lxi     b,buff
4b21 cdbb4b             call    setdma
                ;
                ;       reset monitor entry points
4b24 3ec3               mvi     a,jmp
4b26 320000             sta     0
4b29 21034a             lxi     h,wboote
4b2c 220100             shld    1       ;jmp wboot at location 00
4b2f 320500             sta     5
4b32 21063c             lxi     h,bdos
4b35 220600             shld    6       ;jmp bdos at location 5
4b38 323800             sta     7*8     ;jmp to mon80 (may have been chan
4b3b 2100f8             lxi     h,mon80
4b3e 223900             shld    7*8+1
                ;       leave iobyte set
```

```
                ;       previously selected disk was b, send parameter to
4b41 3a0400             lda     cdisk   ;last logged disk number
4b44 4f                 mov     c,a     ;send to ccp to log it in
4b45 fb                 ei
4b46 c30034             jmp     cpmb
                ;
                ;       error condition occurred, print message and retry
                booterr:
4b49 c1                 pop     b       ;recall counts
4b4a 0d                 dcr     c
4b4b ca524b             jz      booter0
                ;       try again
4b4e c5                 push    b
4b4f c3c94a             jmp     wboot0
                ;
                booter0:
                ;       otherwise too many retries
4b52 215b4b             lxi     h,bootmsg
4b55 cdd34b             call    prmsg
4b58 c30fff             jmp     rmon80  ;mds hardware monitor
                ;
                bootmsg:
4b5b 3f626f4            db      '?boot',0
                ;
                ;
                const:  ;console status to reg-a
                ;       (exactly the same as mds call)
4b61 c312f8             jmp     csts
                ;
                conin:  ;console character to reg-a
4b64 cd03f8             call    ci
4b67 e67f               ani     7fh     ;remove parity bit
4b69 c9                 ret
                ;
                conout: ;console character from c to console out
4b6a c309f8             jmp     co
                ;
                list:   ;list device out
                ;       (exactly the same as mds call)
4b6d c30ff8             jmp     lo
                ;
                listst:
                        ;return list status
4b70 af                 xra     a
4b71 c9                 ret             ;always not ready
                ;
                punch:  ;punch device out
                ;       (exactly the same as mds call)
4b72 c30cf8             jmp     po
                ;
                reader: ;reader character in to reg-a
                ;       (exactly the same as mds call)
4b75 c306f8             jmp     ri
                ;
                home:   ;move to home position
```

```
                ;       treat as track 00 seek
4b78 0e00               mvi     c,0
4b7a c3a74b             jmp     settrk
                ;
                seldsk: ;select disk given by register c
4b7d 210000             lxi     h,0000h ;return 0000 if error
4b80 79                 mov     a,c
4b81 fe04               cpi     ndisks  ;too large?
4b83 d0                 rnc             ;leave hl = 0000
                ;
4b84 e602               ani     10b     ;00 00 for drive 0,1 and 10 10 fo
4b86 32664c             sta     dbank   ;to select drive bank
4b89 79                 mov     a,c     ;00, 01, 10, 11
4b8a e601               ani     1b      ;mds has 0,1 at 78, 2,3 at 88
4b8c b7                 ora     a       ;result 00?
4b8d ca924b             jz      setdrive
4b90 3e30               mvi     a,00110000b     ;selects drive 1 in bank
                setdrive:
4b92 47                 mov     b,a     ;save the function
4b93 21684c             lxi     h,iof   ;io function
4b96 7e                 mov     a,m
4b97 e6cf               ani     11001111b       ;mask out disk number
4b99 b0                 ora     b       ;mask in new disk number
4b9a 77                 mov     m,a     ;save it in iopb
4b9b 69                 mov     l,c
4b9c 2600               mvi     h,0     ;hl=disk number
4b9e 29                 dad     h       ;*2
4b9f 29                 dad     h       ;*4
4ba0 29                 dad     h       ;*8
4ba1 29                 dad     h       ;*16
4ba2 11334a             lxi     d,dpbase
4ba5 19                 dad     d       ;hl=disk header table address
4ba6 c9                 ret
                ;
                ;
                settrk: ;set track address given by c
4ba7 216a4c             lxi     h,iot
4baa 71                 mov     m,c
4bab c9                 ret
                ;
                setsec: ;set sector number given by c
4bac 216b4c             lxi     h,ios
4baf 71                 mov     m,c
4bb0 c9                 ret
                sectran:
                                ;translate sector bc using table at de
4bb1 0600               mvi     b,0     ;double precision sector number i
4bb3 eb                 xchg            ;translate table address to hl
4bb4 09                 dad     b       ;translate(sector) address
4bb5 7e                 mov     a,m     ;translated sector number to a
4bb6 326b4c             sta     ios
4bb9 6f                 mov     l,a     ;return sector number in l
4bba c9                 ret
                ;
                setdma: ;set dma address given by regs b,c
```

```
4bbb 69                 mov     l,c
4bbc 60                 mov     h,b
4bbd 226c4c             shld    iod
4bc0 c9                 ret
              ;
              read:     ;read next disk record (assuming disk/trk/sec/dma
4bc1 0e04               mvi     c,readf ;set to read function
4bc3 cde04b             call    setfunc
4bc6 cdf04b             call    waitio  ;perform read function
4bc9 c9                 ret             ;may have error set in reg-a
              ;
              ;
              write:    ;disk write function
4bca 0e06               mvi     c,writf
4bcc cde04b             call    setfunc ;set to write function
4bcf cdf04b             call    waitio
4bd2 c9                 ret             ;may have error set
              ;
              ;
              ;         utility subroutines
              prmsg:    ;print message at h,l to 0
4bd3 7e                 mov     a,m
4bd4 b7                 ora     a       ;zero?
4bd5 c8                 rz
              ;         more to print
4bd6 e5                 push    h
4bd7 4f                 mov     c,a
4bd8 cd6a4b             call    conout
4bdb e1                 pop     h
4bdc 23                 inx     h
4bdd c3d34b             jmp     prmsg
              ;
              setfunc:
              ;         set function for next i/o (command in reg-c)
4be0 21684c             lxi     h,iof   ;io function address
4be3 7e                 mov     a,m     ;get it to accumulator for maskin
4be4 e6f8               ani     11111000b       ;remove previous command
4be6 b1                 ora     c       ;set to new command
4be7 77                 mov     m,a     ;replaced in iopb
              ;         the mds-800 controller req's disk bank bit in sec
              ;         mask the bit from the current i/o function
4be8 e620               ani     00100000b       ;mask the disk select bit
4bea 216b4c             lxi     h,ios           ;address the sector selec
4bed b6                 ora     m               ;select proper disk bank
4bee 77                 mov     m,a             ;set disk select bit on/o
4bef c9                 ret
              ;
              waitio:
4bf0 0e0a               mvi     c,retry ;max retries before perm error
              rewait:
              ;         start the i/o function and wait for completion
4bf2 cd3f4c             call    intype  ;in rtype
4bf5 cd4c4c             call    inbyte  ;clears the controller
              ;
4bf8 3a664c             lda     dbank           ;set bank flags
```

```
4bfb b7                 ora     a                   ;zero if drive 0,1 and nz
4bfc 3e67               mvi     a,iopb and 0ffh ;low address for iopb
4bfe 064c               mvi     b,iopb shr 8    ;high address for iopb
4c00 c20b4c             jnz     iodrl   ;drive bank 1?
4c03 d379               out     ilow                ;low address to controlle
4c05 78                 mov     a,b
4c06 d37a               out     ihigh   ;high address
4c08 c3104c             jmp     wait0               ;to wait for complete
                ;
                iodrl:  ;drive bank 1
4c0b d389               out     ilow+10h            ;88 for drive bank 10
4c0d 78                 mov     a,b
4c0e d38a               out     ihigh+10h
                ;
4c10 cd594c     wait0:  call    instat              ;wait for completion
4c13 e604               ani     iordy               ;ready?
4c15 ca104c             jz      wait0
                ;
                ;       check io completion ok
4c18 cd3f4c             call    intype              ;must be io complete (00)
                ;       00 unlinked i/o complete,    01 linked i/o comple
                ;       10 disk status changed       11 (not used)
4c1b fe02               cpi     10b                 ;ready status change?
4c1d ca324c             jz      wready
                ;
                ;       must be 00 in the accumulator
4c20 b7                 ora     a
4c21 c2384c             jnz     werror              ;some other condition, re
                ;
                ;       check i/o error bits
4c24 cd4c4c             call    inbyte
4c27 17                 ral
4c28 da324c             jc      wready              ;unit not ready
4c2b 1f                 rar
4c2c e6fe               ani     11111110b           ;any other errors?
4c2e c2384c             jnz     werror
                ;
                ;       read or write is ok, accumulator contains zero
4c31 c9                 ret
                ;
                wready: ;not ready, treat as error for now
4c32 cd4c4c             call    inbyte              ;clear result byte
4c35 c3384c             jmp     trycount
                ;
                werror: ;return hardware malfunction (crc, track, seek, e
                ;       the mds controller has returned a bit in each pos
                ;       of the accumulator, corresponding to the conditio
                ;       0       - deleted data (accepted as ok above)
                ;       1       - crc error
                ;       2       - seek error
                ;       3       - address error (hardware malfunction)
                ;       4       - data over/under flow (hardware malfunct
                ;       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 condit
                ;       but we will get a permanent error message if it i
                ;       recoverable.  in any case, the not ready conditio
                ;       treated as a separate condition for later improve
                trycount:
                ;       register c contains retry count, decrement 'til z
4c38 0d                 dcr     c
4c39 c2f24b             jnz     rewait  ;for another try
                ;
                ;       cannot recover from error
4c3c 3e01               mvi     a,1     ;error code
4c3e c9                 ret
                ;
                ;       intype, inbyte, instat read drive bank 00 or 10
4c3f 3a664c     intype: lda     dbank
4c42 b7                 ora     a
4c43 c2494c             jnz     intypl  ;skip to bank 10
4c46 db79               in      rtype
4c48 c9                 ret
4c49 db89       intypl: in      rtype+10h       ;78 for 0,1  88 for 2,3
4c4b c9                 ret
                ;
4c4c 3a664c     inbyte: lda     dbank
4c4f b7                 ora     a
4c50 c2564c             jnz     inbytl
4c53 db7b               in      rbyte
4c55 c9                 ret
4c56 db8b       inbytl: in      rbyte+10h
4c58 c9                 ret
                ;
4c59 3a664c     instat: lda     dbank
4c5c b7                 ora     a
4c5d c2634c             jnz     instal
4c60 db78               in      dstat
4c62 c9                 ret
4c63 db88       instal: in      dstat+10h
4c65 c9                 ret
                ;
                ;
                ;
                ;       data areas (must be in ram)
4c66 00         dbank:  db      0       ;disk bank 00 if drive 0,1
                                        ;          10 if drive 2,3
                iopb:   ;io parameter block
4c67 80                 db      80h     ;normal i/o operation
4c68 04         iof:    db      readf   ;io function, initial read
4c69 01         ion:    db      1       ;number of sectors to read
4c6a 02         iot:    db      offset  ;track number
4c6b 01         ios:    db      1       ;sector number
4c6c 8000       iod:    dw      buff    ;io address
                ;
                ;
                ;       define ram areas for bdos operation
```

```
                        endef
4c6e+=      begdat      equ     $
4c6e+       dirbuf:     ds      128     ;directory access buffer
4cee+       alvØ:       ds      31
4dØd+       csvØ:       ds      16
4d1d+       alv1:       ds      31
4d3c+       csv1:       ds      16
4d4c+       alv2:       ds      31
4d6b+       csv2:       ds      16
4d7b+       alv3:       ds      31
4d9a+       csv3:       ds      16
4daa+=      enddat      equ     $
Ø13c+=      datsiz      equ     $-begdat
4daa                    end
```

APPENDIX C: A SKELETAL CBIOS

```
                ;       skeletal cbios for first level of cp/m 2.0 altera
                ;
0014 =          msize   equ     20      ;cp/m version memory size in kilo
                ;
                ;       "bias" is address offset from 3400h for memory sy
                ;       than 16k (referred to as "b" throughout the text)
                ;
0000 =          bias    equ     (msize-20)*1024
3400 =          ccp     equ     3400h+bias      ;base of ccp
3c06 =          bdos    equ     ccp+806h        ;base of bdos
4a00 =          bios    equ     ccp+1600h       ;base of bios
0004 =          cdisk   equ     0004h   ;current disk number 0=a,...,15=p
0003 =          iobyte  equ     0003h   ;intel i/o byte
                ;
4a00                    org     bios    ;origin of this program
002c =          nsects  equ     ($-ccp)/128     ;warm start sector count
                ;
                ;       jump vector for individual subroutines
4a00 c39c4a             jmp     boot            ;cold start
4a03 c3a64a     wboote: jmp     wboot           ;warm start
4a06 c3114b             jmp     const           ;console status
4a09 c3244b             jmp     conin           ;console character in
4a0c c3374b             jmp     conout          ;console character out
4a0f c3494b             jmp     list            ;list character out
4a12 c34d4b             jmp     punch           ;punch character out
4a15 c34f4b             jmp     reader          ;reader character out
4a18 c3544b             jmp     home            ;move head to home positi
4a1b c35a4b             jmp     seldsk          ;select disk
4a1e c37d4b             jmp     settrk          ;set track number
4a21 c3924b             jmp     setsec          ;set sector number
4a24 c3ad4b             jmp     setdma          ;set dma address
4a27 c3c34b             jmp     read            ;read disk
4a2a c3d64b             jmp     write           ;write disk
4a2d c34b4b             jmp     listst          ;return list status
4a30 c3a74b             jmp     sectran         ;sector translate
                ;
                ;       fixed data tables for four-drive standard
                ;       ibm-compatible 8" disks
                ;       disk parameter header for disk 00
4a33 734a00     dpbase: dw      trans,0000h
4a37 000000             dw      0000h,0000h
4a3b f04c8d             dw      dirbf,dpblk
4a3f ec4d70             dw      chk00,all00
                ;       disk parameter header for disk 01
4a43 734a00             dw      trans,0000h
4a47 000000             dw      0000h,0000h
4a4b f04c8d             dw      dirbf,dpblk
4a4f fc4d8f             dw      chk01,all01
                ;       disk parameter header for disk 02
4a53 734a00             dw      trans,0000h
4a57 000000             dw      0000h,0000h
4a5b f04c8d             dw      dirbf,dpblk
4a5f 0c4eae             dw      chk02,all02
```

```
                ;       disk parameter header for disk 03
4a63 734a00             dw      trans,0000h
4a67 000000             dw      0000h,0000h
4a6b f04c8d             dw      dirbf,dpblk
4a6f 1c4ecd             dw      chk03,all03
                ;
                ;       sector translate vector
4a73 01070d     trans:  db      1,7,13,19           ;sectors 1,2,3,4
4a77 19050b             db      25,5,11,17          ;sectors 5,6,7,8
4a7b 170309             db      23,3,9,15           ;sectors 9,10,11,12
4a7f 150208             db      21,2,8,14           ;sectors 13,14,15,16
4a83 141a06             db      20,26,6,12          ;sectors 17,18,19,20
4a87 121804             db      18,24,4,10          ;sectors 21,22,23,24
4a8b 1016               db      16,22               ;sectors 25,26
                ;
                dpblk:  ;disk parameter block, common to all disks
4a8d 1a00               dw      26                  ;sectors per track
4a8f 03                 db      3                   ;block shift factor
4a90 07                 db      7                   ;block mask
4a91 00                 db      0                   ;null mask
4a92 f200               dw      242                 ;disk size-1
4a94 3f00               dw      63                  ;directory max
4a96 c0                 db      192                 ;alloc 0
4a97 00                 db      0                   ;alloc 1
4a98 1000               dw      16                  ;check size
4a9a 0200               dw      2                   ;track offset
                ;
                ;       end of fixed tables
                ;
                ;       individual subroutines to perform each function
                boot:   ;simplest case is to just perform parameter initi
4a9c af                 xra     a                   ;zero in the accum
4a9d 320300             sta     iobyte              ;clear the iobyte
4aa0 320400             sta     cdisk               ;select disk zero
4aa3 c3ef4a             jmp     gocpm               ;initialize and go to cp/
                ;
                wboot:  ;simplest case is to read the disk until all sect
4aa6 318000             lxi     sp,80h              ;use space below buffer f
4aa9 0e00               mvi     c,0                 ;select disk 0
4aab cd5a4b             call    seldsk
4aae cd544b             call    home                ;go to track 00
                ;
4ab1 062c               mvi     b,nsects            ;b counts # of sectors to
4ab3 0e00               mvi     c,0                 ;c has the current track
4ab5 1602               mvi     d,2                 ;d has the next sector to
                ;       note that we begin by reading track 0, sector 2 s
                ;       contains the cold start loader, which is skipped
4ab7 210034             lxi     h,ccp               ;base of cp/m (initial lo
                load1:  ;load one more sector
4aba c5                 push    b       ;save sector count, current track
4abb d5                 push    d       ;save next sector to read
4abc e5                 push    h       ;save dma address
4abd 4a                 mov     c,d     ;get sector address to register c
4abe cd924b             call    setsec  ;set sector address from register
4ac1 c1                 pop     b       ;recall dma address to b,c
```

```
4ac2 c5                 push    b       ;replace on stack for later recal
4ac3 cdad4b             call    setdma  ;set dma address from b,c
                ;
                ;       drive set to Ø, track set, sector set, dma addres
4ac6 cdc34b             call    read
4ac9 feØØ               cpi     ØØh     ;any errors?
4acb c2a64a             jnz     wboot   ;retry the entire boot if an erro
                ;
                ;       no error, move to next sector
4ace el                 pop     h       ;recall dma address
4acf 118ØØØ             lxi     d,128   ;dma=dma+128
4ad2 19                 dad     d       ;new dma address is in h,l
4ad3 dl                 pop     d       ;recall sector address
4ad4 cl                 pop     b       ;recall number of sectors remaini
4ad5 Ø5                 dcr     b       ;sectors=sectors-1
4ad6 caef4a             jz      gocpm   ;transfer to cp/m if all have bee
                ;
                ;       more sectors remain to load, check for track chan
4ad9 14                 inr     d
4ada 7a                 mov     a,d     ;sector=27?, if so, change tracks
4adb felb               cpi     27
4add daba4a             jc      loadl   ;carry generated if sector<27
                ;
                ;       end of current track, go to next track
4aeØ 16Ø1               mvi     d,l     ;begin with first sector of next
4ae2 Øc                 inr     c       ;track=track+1
                ;
                ;       save register state, and change tracks
4ae3 c5                 push    b
4ae4 d5                 push    d
4ae5 e5                 push    h
4ae6 cd7d4b             call    settrk  ;track address set from register
4ae9 el                 pop     h
4aea dl                 pop     d
4aeb cl                 pop     b
4aec c3ba4a             jmp     loadl   ;for another sector
                ;
                ;       end of load operation, set parameters and go to c
                gocpm:
4aef 3ec3               mvi     a,Øc3h  ;c3 is a jmp instruction
4afl 32ØØØØ             sta     Ø       ;for jmp to wboot
4af4 21Ø34a             lxi     h,wboote        ;wboot entry point
4af7 22Ø1ØØ             shld    l       ;set address field for jmp at Ø
                ;
4afa 32Ø5ØØ             sta     5       ;for jmp to bdos
4afd 21Ø63c             lxi     h,bdos  ;bdos entry point
4bØØ 22Ø6ØØ             shld    6       ;address field of jump at 5 to bd
                ;
4bØ3 Ø18ØØØ             lxi     b,8Øh   ;default dma address is 8Øh
4bØ6 cdad4b             call    setdma
                ;
4bØ9 fb                 ei              ;enable the interrupt system
4bØa 3aØ4ØØ             lda     cdisk   ;get current disk number
4bØd 4f                 mov     c,a     ;send to the ccp
4bØe c3ØØ34             jmp     ccp     ;go to cp/m for further processin
```

```
                ;
                ;
                ;       simple i/o handlers (must be filled in by user)
                ;       in each case, the entry point is provided, with s
                ;       to insert your own code
                ;
                const:  ;console status, return 0ffh if character ready,
4b11                    ds      10h     ;space for status subroutine
4b21 3e00               mvi     a,00h
4b23 c9                 ret
                ;
                conin:  ;console character into register a
4b24                    ds      10h     ;space for input routine
4b34 e67f               ani     7fh     ;strip parity bit
4b36 c9                 ret
                ;
                conout: ;console character output from register c
4b37 79                 mov     a,c     ;get to accumulator
4b38                    ds      10h     ;space for output routine
4b48 c9                 ret
                ;
                list:   ;list character from register c
4b49 79                 mov     a,c     ;character to register a
4b4a c9                 ret             ;null subroutine
                ;
                listst: ;return list status (0 if not ready, 1 if ready)
4b4b af                 xra     a       ;0 is always ok to return
4b4c c9                 ret
                ;
                punch:  ;punch character from register c
4b4d 79                 mov     a,c     ;character to register a
4b4e c9                 ret             ;null subroutine
                ;
                ;
                reader: ;read character into register a from reader devic
4b4f 3e1a               mvi     a,1ah   ;enter end of file for now (repla
4b51 e67f               ani     7fh     ;remember to strip parity bit
4b53 c9                 ret
                ;
                ;
                ;       i/o drivers for the disk follow
                ;       for now, we will simply store the parameters away
                ;       in the read and write subroutines
                ;
                home:   ;move to the track 00 position of current drive
                ;       translate this call into a settrk call with param
4b54 0e00               mvi     c,0     ;select track 0
4b56 cd7d4b             call    settrk
4b59 c9                 ret             ;we will move to 00 on first read
                ;
                seldsk: ;select disk given by register c
4b5a 210000             lxi     h,0000h ;error return code
4b5d 79                 mov     a,c
4b5e 32ef4c             sta     diskno
4b61 fe04               cpi     4       ;must be between 0 and 3
```

```
4b63 d0            rnc                 ;no carry if 4,5,...
          ;        disk number is in the proper range
4b64               ds       10         ;space for disk select
          ;        compute proper disk parameter header address
4b6e 3aef4c        lda      diskno
4b71 6f            mov      l,a        ;l=disk number 0,1,2,3
4b72 2600          mvi      h,0        ;high order zero
4b74 29            dad      h          ;*2
4b75 29            dad      h          ;*4
4b76 29            dad      h          ;*8
4b77 29            dad      h          ;*16 (size of each header)
4b78 11334a        lxi      d,dpbase
4b7b 19            dad      d          ;hl=.dpbase(diskno*16)
4b7c c9            ret
          ;
          settrk:  ;set track given by register c
4b7d 79            mov      a,c
4b7e 32e94c        sta      track
4b81               ds       10h        ;space for track select
4b91 c9            ret
          ;
          setsec:  ;set sector given by register c
4b92 79            mov      a,c
4b93 32eb4c        sta      sector
4b96               ds       10h        ;space for sector select
4ba6 c9            ret
          ;
          sectran:
                   ;translate the sector given by bc using the
                   ;translate table given by de
4ba7 eb            xchg                ;hl=.trans
4ba8 09            dad      b          ;hl=.trans(sector)
4ba9 6e            mov      l,m        ;l = trans(sector)
4baa 2600          mvi      h,0        ;hl= trans(sector)
4bac c9            ret                 ;with value in hl
          ;
          setdma:  ;set dma address given by registers b and c
4bad 69            mov      l,c        ;low order address
4bae 60            mov      h,b        ;high order address
4baf 22ed4c        shld     dmaad      ;save the address
4bb2               ds       10h        ;space for setting the dma addres
4bc2 c9            ret
          ;
          read:    ;perform read operation (usually this is similar
          ;        so we will allow space to set up read command, th
          ;        common code in write)
4bc3               ds       10h        ;set up read command
4bd3 c3e64b        jmp      waitio     ;to perform the actual i/o
          ;
          write:   ;perform a write operation
4bd6               ds       10h        ;set up write commanu
          ;
          waitio:  ;enter here from read and write to perform the ac
          ;        operation.  return a 00h in register a if the ope
          ;        properly, and 01h if an error occurs during the r
```

```
                ;
                ;       in this case, we have saved the disk number in 'd
                ;                       the track number in 'track' (Ø-76
                ;                       the sector number in 'sector' (1-
                ;                       the dma address in 'dmaad' (Ø-655
4be6                    ds      256     ;space reserved for i/o drivers
4ce6 3eØ1               mvi     a,1     ;error condition
4ce8 c9                 ret             ;replaced when filled-in
                ;
                ;       the remainder of the cbios is reserved uninitiali
                ;       data area, and does not need to be a part of the
                ;       system memory image (the space must be available,
                ;       however, between "begdat" and "enddat").
                ;
4ce9            track:  ds      2       ;two bytes for expansion
4ceb            sector: ds      2       ;two bytes for expansion
4ced            dmaad:  ds      2       ;direct memory address
4cef            diskno: ds      1       ;disk number Ø-15
                ;
                ;       scratch ram area for bdos use
4cfØ =          begdat  equ     $       ;beginning of data area
4cfØ            dirbf:  ds      128     ;scratch directory area
4d7Ø            all ØØ: ds      31      ;allocation vector Ø
4d8f            all Ø1: ds      31      ;allocation vector 1
4dae            all Ø2: ds      31      ;allocation vector 2
4dcd            all Ø3: ds      31      ;allocation vector 3
4dec            chkØØ:  ds      16      ;check vector Ø
4dfc            chkØ1:  ds      16      ;check vector 1
4eØc            chkØ2:  ds      16      ;check vector 2
4e1c            chkØ3:  ds      16      ;check vector 3
                ;
4e2c =          enddat  equ     $       ;end of data area
Ø13c =          datsiz  equ     $-begdat;size of data area
4e2c                    end
```

APPENDIX D: A SKELETAL GETSYS/PUTSYS PROGRAM

```
                ;       combined getsys and putsys programs from Sec 4.
                ;       Start the programs at the base of the TPA

0100                    org     0100h

0014 =          msize   equ     20                ; size of cp/m in Kbytes

                ; "bias" is the amount to add to addresses for > 20k
                ;       (referred to as "b" throughout the text)

0000 =          bias    equ     (msize-20)*1024
3400 =          ccp     equ     3400h+bias
3c00 =          bdos    equ     ccp+0800h
4a00 =          bios    equ     ccp+1600h

                ;       getsys programs tracks 0 and 1 to memory at
                ;       3880h + bias

                ;       register                usage
                ;         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:                                 ; start of getsys
0100 318033             lxi     sp,ccp-0080h            ; convenient plac
0103 218033             lxi     h,ccp-0080h             ; set initial loa
0106 0600               mvi     b,0                     ; start with trac
                rd$trk:                                 ; read next track
0108 0e01               mvi     c,1                     ; each track star
                rd$sec:
010a cd0003             call    read$sec                ; get the next se
010d 118000             lxi     d,128                   ; offset by one s
0110 19                 dad     d                       ;   (hl=hl+128)
0111 0c                 inr     c                       ; next sector
0112 79                 mov     a,c                     ; fetch sector nu
0113 fe1b               cpi     27                      ;   and see if la
0115 da0a01             jc      rdsec                   ; <, do one more

                ; 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
011a fe02               cpi     2                       ; track = 2 ?
011c da0801             jc      rd$trk                  ; <, do another

                ; arrive here at end of load, halt for lack of anything b

011f fb                 ei
0120 76                 hlt
```

```
                ;       putsys program, places memory image starting at
                ;       3880h + bias back to tracks 0 and 1
                ;       start this program at the next page boundary

0200                    org     ($+0100h) and 0ff00h

                put$sys:
0200 318033             lxi     sp,ccp-0080h            ; convenient plac
0203 218033             lxi     h,ccp-0080h             ; start of dump
0206 0600               mvi     b,0                     ; start with trac
                wr$trk:
0208 0e01               mvi     c,1                     ; start with sect
                wr$sec:
020a cd0004             call    write$sec               ; write one secto
020d 118000             lxi     d,128                   ; length of each
0210 19                 dad     d                       ; <hl>=<hl> + 128
0211 0c                 inr     c                       ; <c> = <c> + 1
0212 79                 mov     a,c                     ; see if
0213 fe1b               cpi     27                      ;   past end of t
0215 da0a02             jc      wr$sec                  ; no, do another

                ;  arrive here at end of track, move to next track

0218 04                 inr     b                       ; track = track+1
0219 78                 mov     a,b                     ; see if
021a fe02               cpi     2                       ;   last track
021c da0802             jc      wr$trk                  ; no, do another

                ;       done with putsys, halt for lack of anything bette

021f fb                 ei
0220 76                 hlt


                ; user supplied subroutines for sector read and write

                ;       move to next page boundary

0300                    org     ($+0100h) and 0ff00h

                read$sec:
                        ; read the next sector
                        ; track in <b>,
                        ; sector in <c>
                        ; dmaaddr in <hl>

0300 c5                 push    b
0301 e5                 push    h

                ; user defined read operation goes here
0302                    ds      64

0342 e1                 pop     h
0343 c1                 pop     b
```

```
0344 c9              ret

0400                 org     ($+0100h) and 0ff00h    ; another page bo

           write$sec:

                     ; same parameters as read$sec

0400 c5              push    b
0401 e5              push    h

           ; user defined write operation goes here
0402                 ds      64

0442 el              pop     h
0443 cl              pop     b
0444 c9              ret

           ; end of getsys/putsys program

0445                 end
```

APPENDIX E: 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 start-up (this pro-
                ; gram can be keyed-in, or can exist in 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" (3400h + "bias").  in a 20k
                ; memory system, the value of "bias" is 0000h, with large
                ; values for increased memory sizes (see section 2). afte
                ; loading the cp/m system, the clod start loader branches
                ; to the "boot" entry point of the bios, which begins at
                ; "bios" + "bias."  the cold start loader is not used un-
                ; til the system is powered up again, as long as the bios
                ; is not overwritten.  the origin is assumed at 0000h, an
                ; must be changed if the controller brings the cold start
                ; loader into another area, or if a read/only memory area
                ; is used.

0000                    org     0               ; base of ram in cp/m

0014 =          msize   equ     20              ; min mem size in kbytes

0000 =          bias    equ     (msize-20)*1024 ; offset from 20k system
3400 =          ccp     equ     3400h+bias      ; base of the ccp
4a00 =          bios    equ     ccp+1600h       ; base of the bios
0300 =          biosl   equ     0300h           ; length of the bios
4a00 =          boot    equ     bios
1900 =          size    equ     bios+biosl-ccp  ; size of cp/m system
0032 =          sects   equ     size/128        ; # of sectors to load

                ;       begin the load operation

                cold:
0000 010200             lxi     b,2             ; b=0, c=sector 2
0003 1632               mvi     d,sects         ; d=# sectors to load
0005 210034             lxi     h,ccp           ; base transfer address

                lsect:  ; load the next sector

                ;       insert inline code at this point to
                ;       read one 128 byte sector from the
                ;       track given in register b, sector
                ;       given in register c,
                ;       into the address given by <hl>
                ;
                ; branch to location "cold" if a read error occurs
```

```
                ;       ************************************************
                ;       *
                ;       *       user supplied read operation goes here...
                ;       *
                ;       ************************************************

0008 c36b00             jmp     past$patch      ; remove this when patche
000b                    ds      60h

                past$patch:
                ; go to next sector if load is incomplete
006b 15                 dcr     d               ; sects=sects-1
006c ca004a             jz      boot            ; head for the bios

                ;       more sectors to load
                ;
                ; we aren't using a stack, so use <sp> as scratch registe
                ;       to hold the load address increment

006f 318000             lxi     sp,128          ; 128 bytes per sector
0072 39                 dad     sp              ; <hl> = <hl> + 128

0073 0c                 inr     c               ; sector = sector + 1
0074 79                 mov     a,c
0075 felb               cpi     27              ; last sector of track?
0077 da0800             jc      lsect           ; no, go read another

                ; end of track, increment to next track

007a 0e01               mvi     c,1             ; sector = 1
007c 04                 inr     b               ; track = track + 1
007d c30800             jmp     lsect           ; for another group
0080                    end                     ; of boot loader
```

APPENDIX F: CP/M DISK DEFINITION LIBRARY

```
;       CP/M 2.0 disk re-definition library
;
;       Copyright (c) 1979
;       Digital Research
;       Box 579
;       Pacific Grove, CA
;       93950
;
;       CP/M logical disk drives are defined using the
;       macros given below, where the sequence of calls
;       is:
;
;       disks   n
;       diskdef parameter-list-0
;       diskdef parameter-list-1
;       ...
;       diskdef parameter-list-n
;       endef
;
;       where n is the number of logical disk drives attached
;       to the CP/M system, and parameter-list-i defines the
;       characteristics of the ith drive (i=0,1,...,n-1)
;
;       each parameter-list-i takes the form
;               dn,fsc,lsc,[skf],bls,dks,dir,cks,ofs,[0]
;       where
;       dn      is the disk number 0,1,...,n-1
;       fsc     is the first sector number (usually 0 or 1)
;       lsc     is the last sector number on a track
;       skf     is optional "skew factor" for sector translate
;       bls     is the data block size (1024,2048,...,16384)
;       dks     is the disk size in bls increments (word)
;       dir     is the number of directory elements (word)
;       cks     is the number of dir elements to checksum
;       ofs     is the number of tracks to skip (word)
;       [0]     is an optional 0 which forces 16K/directory en
;
;       for convenience, the form
;               dn,dm
;       defines disk dn as having the same characteristics as
;       a previously defined disk dm.
;
;       a standard four drive CP/M system is defined by
;               disks   4
;               diskdef 0,1,26,6,1024,243,64,64,2
;       dsk     set     0
;               rept    3
;       dsk     set     dsk+1
;               diskdef %dsk,0
;               endm
;               endef
;
;       the value of "begdat" at the end of assembly defines t
```

```
;        beginning of the uninitialize ram area above the bios,
;        while the value of "enddat" defines the next location
;        following the end of the data area.  the size of this
;        area is given by the value of "datsiz" at the end of t
;        assembly.  note that the allocation vector will be qui
;        large if a large disk size is defined with a small blo
;        size.
;
dskhdr   macro   dn
;;       define a single disk header list
dpe&dn:  dw      xlt&dn,0000h     ;translate table
         dw      0000h,0000h      ;scratch area
         dw      dirbuf,dpb&dn    ;dir buff,parm block
         dw      csv&dn,alv&dn    ;check, alloc vectors
         endm
;
disks    macro   nd
;;       define nd disks
ndisks   set     nd       ;;for later reference
dpbase   equ     $        ;base of disk parameter blocks
;;       generate the nd elements
dsknxt   set     0
         rept    nd
         dskhdr  %dsknxt
dsknxt   set     dsknxt+1
         endm
         endm
;
dpbhdr   macro   dn
dpb&dn   equ     $                ;disk parm block
         endm
;
ddb      macro   data,comment
;;       define a db statement
         db      data             comment
         endm
;
ddw      macro   data,comment
;;       define a dw statement
         dw      data             comment
         endm
;
gcd      macro   m,n
;;       greatest common divisor of m,n
;;       produces value gcdn as result
;;       (used in sector translate table generation)
gcdm     set     m        ;;variable for m
gcdn     set     n        ;;variable for n
gcdr     set     0        ;;variable for r
         rept    65535
gcdx     set     gcdm/gcdn
gcdr     set     gcdm - gcdx*gcdn
         if      gcdr = 0
         exitm
         endif
```

```
gcdm    set     gcdn
gcdn    set     gcdr
        endm
        endm
;
diskdef macro   dn,fsc,lsc,skf,bls,dks,dir,cks,ofs,k16
;;      generate the set statements for later tables
        if      nul lsc
;;      current disk dn same as previous fsc
dpb&dn  equ     dpb&fsc ;equivalent parameters
als&dn  equ     als&fsc ;same allocation vector size
css&dn  equ     css&fsc ;same checksum vector size
xlt&dn  equ     xlt&fsc ;same translate table
        else
secmax  set     lsc-(fsc)       ;;sectors 0...secmax
sectors set     secmax+1;;number of sectors
als&dn  set     (dks)/8 ;;size of allocation vector
        if      ((dks) mod 8) ne 0
als&dn  set     als&dn+1
        endif
css&dn  set     (cks)/4 ;;number of checksum elements
;;      generate the block shift value
blkval  set     bls/128 ;;number of sectors/block
blkshf  set     0       ;;counts right 0's in blkval
blkmsk  set     0       ;;fills with 1's from right
        rept    16      ;;once for each bit position
        if      blkval=1
        exitm
        endif
;;      otherwise, high order 1 not found yet
blkshf  set     blkshf+1
blkmsk  set     (blkmsk shl 1) or 1
blkval  set     blkval/2
        endm
;;      generate the extent mask byte
blkval  set     bls/1024        ;;number of kilobytes/block
extmsk  set     0       ;;fill from right with 1's
        rept    16
        if      blkval=1
        exitm
        endif
;;      otherwise more to shift
extmsk  set     (extmsk shl 1) or 1
blkval  set     blkval/2
        endm
;;      may be double byte allocation
        if      (dks) > 256
extmsk  set     (extmsk shr 1)
        endif
;;      may be optional [0] in last position
        if      not nul k16
extmsk  set     k16
        endif
;;      now generate directory reservation bit vector
dirrem  set     dir     ;;# remaining to process
```

```
dirbks  set     bls/32  ;;number of entries per block
dirblk  set     0       ;;fill with 1's on each loop
        rept    16
        if      dirrem=0
        exitm
        endif
;;      not complete, iterate once again
;;      shift right and add 1 high order bit
dirblk  set     (dirblk shr 1) or 8000h
        if      dirrem > dirbks
dirrem  set     dirrem-dirbks
        else
dirrem  set     0
        endif
        endm
        dpbhdr  dn      ;;generate equ $
        ddw     %sectors,<;sec per track>
        ddb     %blkshf,<;block shift>
        ddb     %blkmsk,<;block mask>
        ddb     %extmsk,<;extnt mask>
        ddw     %(dks)-1,<;disk size-1>
        ddw     %(dir)-1,<;directory max>
        ddb     %dirblk shr 8,<;alloc0>
        ddb     %dirblk and 0ffh,<;alloc1>
        ddw     %(cks)/4,<;check size>
        ddw     %ofs,<;offset>
;;      generate the translate table, if requested
        if      nul skf
xlt&dn  equ     0                   ;no xlate table
        else
        if      skf = 0
xlt&dn  equ     0                   ;no xlate table
        else
;;      generate the translate table
nxtsec  set     0       ;;next sector to fill
nxtbas  set     0       ;;moves by one on overflow
        gcd     %sectors,skf
;;      gcdn = gcd(sectors,skew)
neltst  set     sectors/gcdn
;;      neltst is number of elements to generate
;;      before we overlap previous elements
nelts   set     neltst  ;;counter
xlt&dn  equ     $                   ;translate table
        rept    sectors ;;once for each sector
        if      sectors < 256
        ddb     %nxtsec+(fsc)
        else
        ddw     %nxtsec+(fsc)
        endif
nxtsec  set     nxtsec+(skf)
        if      nxtsec >= sectors
nxtsec  set     nxtsec-sectors
        endif
nelts   set     nelts-1
        if      nelts = 0
```

```
nxtbas  set     nxtbas+1
nxtsec  set     nxtbas
nelts   set     neltst
        endif
        endm
        endif   ;;end of nul fac test
        endif   ;;end of nul bls test
        endm
;
defds   macro   lab,space
lab:    ds      space
        endm
;
lds     macro   lb,dn,val
        defds   lb&dn,%val&dn
        endm
;
endef   macro
;;      generate the necessary ram data areas
begdat  equ     $
dirbuf: ds      128     ;directory access buffer
dsknxt  set     0
        rept    ndisks  ;;once for each disk
        lds     alv,%dsknxt,als
        lds     csv,%dsknxt,css
dsknxt  set     dsknxt+1
        endm
enddat  equ     $
datsiz  equ     $-begdat
;;      db 0 at this point forces hex record
        endm
```

APPENDIX G: BLOCKING AND DEBLOCKING ALGORITHMS.

```
;*****************************************************
;*                                                   *
;*      Sector Deblocking Algorithms for CP/M 2.0    *
;*                                                   *
;*****************************************************
;
;       utility macro to compute sector mask
smask   macro   hblk
;;      compute log2(hblk), return @x as result
;;      (2 ** @x = hblk on return)
@y      set     hblk
@x      set     0
;;      count right shifts of @y until = 1
        rept    8
        if      @y = 1
        exitm
        endif
;;      @y is not 1, shift right one position
@y      set     @y shr 1
@x      set     @x + 1
        endm
        endm
;
;*****************************************************
;*                                                   *
;*         CP/M to host disk constants               *
;*                                                   *
;*****************************************************
blksiz  equu    2048            ;CP/M allocation size
hstsiz  equ     512             ;host disk sector size
hstspt  equ     20              ;host disk sectors/trk
hstblk  equ     hstsiz/128      ;CP/M sects/host buff
cpmspt  equ     hstblk * hstspt ;CP/M sectors/track
secmsk  equ     hstblk-1        ;sector mask
        smask   hstblk          ;compute sector mask
secshf  equ     @x              ;log2(hstblk)
;
;*****************************************************
;*                                                   *
;*         BDOS constants on entry to write          *
;*                                                   *
;*****************************************************
wrall   equ     0               ;write to allocated
wrdir   equ     1               ;write to directory
wrual   equ     2               ;write to unallocated
;
;*****************************************************
;*                                                   *
;*      The BDOS entry points given below show the   *
;*      code which is relevant to deblocking only.   *
;*                                                   *
;*****************************************************
;
```

```
;       DISKDEF macro, or hand coded tables go here
dpbase  equ     $               ;disk param block base
;
boot:
wboot:
        ;enter here on system boot to initialize
        xra     a               ;0 to accumulator
        sta     hstact          ;host buffer inactive
        sta     unacnt          ;clear unalloc count
        ret
;
seldsk:
        ;select disk
        mov     a,c             ;selected disk number
        sta     sekdsk          ;seek disk number
        mov     l,a             ;disk number to HL
        mvi     h,0
        rept    4               ;multiply by 16
        dad     h
        endm
        lxi     d,dpbase        ;base of parm block
        dad     d               ;hl=.dpb(curdsk)
        ret
;
settrk:
        ;set track given by registers BC
        mov     h,b
        mov     l,c
        shld    sektrk          ;track to seek
        ret
;
setsec:
        ;set sector given by register c
        mov     a,c
        sta     seksec          ;sector to seek
        ret
;
setdma:
        ;set dma address given by BC
        mov     h,b
        mov     l,c
        shld    dmaadr
        ret
;
sectran:
        ;translate sector number BC
        mov     h,b
        mov     l,c
        ret
;
```

```
;*******************************************************
;*                                                     *
;*      The READ entry point takes the place of        *
;*      the previous BIOS defintion for READ.          *
;*                                                     *
;*******************************************************
read:
        ;read the selected CP/M sector
        mvi     a,1
        sta     readop          ;read operation
        sta     rsflag          ;must read data
        mvi     a,wrual
        sta     wrtype          ;treat as unalloc
        jmp     rwoper          ;to perform the read
;
;*******************************************************
;*                                                     *
;*      The WRITE entry point takes the place of       *
;*      the previous BIOS defintion for WRITE.         *
;*                                                     *
;*******************************************************
write:
        ;write the selected CP/M sector
        xra     a               ;0 to accumulator
        sta     readop          ;not a read operation
        mov     a,c             ;write type in c
        sta     wrtype
        cpi     wrual           ;write unallocated?
        jnz     chkuna          ;check for unalloc
;
;       write to unallocated, set parameters
        mvi     a,blksiz/128    ;next unalloc recs
        sta     unacnt
        lda     sekdsk          ;disk to seek
        sta     unadsk          ;unadsk = sekdsk
        lhld    sektrk
        shld    unatrk          ;unatrk = sectrk
        lda     seksec
        sta     unasec          ;unasec = seksec
;
chkuna:
        ;check for write to unallocated sector
        lda     unacnt          ;any unalloc remain?
        ora     a
        jz      alloc           ;skip if not
;
;       more unallocated records remain
        dcr     a               ;unacnt = unacnt-1
        sta     unacnt
        lda     sekdsk          ;same disk?
        lxi     h,unadsk
        cmp     m               ;sekdsk = unadsk?
        jnz     alloc           ;skip if not
;
;       disks are the same
```

```
        lxi     h,unatrk
        call    sektrkcmp       ;sektrk = unatrk?
        jnz     alloc           ;skip if not
;
;       tracks are the same
        lda     seksec          ;same sector?
        lxi     h,unasec
        cmp     m               ;seksec = unasec?
        jnz     alloc           ;skip if not
;
;       match, move to next sector for future ref
        inr     m               ;unasec = unasec+1
        mov     a,m             ;end of track?
        cpi     cpmspt          ;count CP/M sectors
        jc      noovf           ;skip if no overflow
;
;       overflow to next track
        mvi     m,0             ;unasec = 0
        lhld    unatrk
        inx     h
        shld    unatrk          ;unatrk = unatrk+1
;
noovf:
        ;match found, mark as unnecessary read
        xra     a               ;0 to accumulator
        sta     rsflag          ;rsflag = 0
        jmp     rwoper          ;to perform the write
;
alloc:
        ;not an unallocated record, requires pre-read
        xra     a               ;0 to accum
        sta     unacnt          ;unacnt = 0
        inr     a               ;1 to accum
        sta     rsflag          ;rsflag = 1
;
;*****************************************************
;*                                                   *
;*      Common code for READ and WRITE follows       *
;*                                                   *
;*****************************************************
rwoper:
        ;enter here to perform the read/write
        xra     a               ;zero to accum
        sta     erflag          ;no errors (yet)
        lda     seksec          ;compute host sector
        rept    secshf
        ora     a               ;carry = 0
        rar                     ;shift right
        endm
        sta     sekhst          ;host sector to seek
;
;       active host sector?
        lxi     h,hstact        ;host active flag
        mov     a,m
        mvi     m,1             ;always becomes 1
```

```
        ora     a               ;was it already?
        jz      filhst          ;fill host if not
;
;       host buffer active, same as seek buffer?
        lda     sekdsk
        lxi     h,hstdsk        ;same disk?
        cmp     m               ;sekdsk = hstdsk?
        jnz     nomatch
;
;       same disk, same track?
        lxi     h,hsttrk
        call    sektrkcmp       ;sektrk = hsttrk?
        jnz     nomatch
;
;       same disk, same track, same buffer?
        lda     sekhst
        lxi     h,hstsec        ;sekhst = hstsec?
        cmp     m
        jz      match           ;skip if match
;
nomatch:
        ;proper disk, but not correct sector
        lda     hstwrt          ;host written?
        ora     a
        cnz     writehst        ;clear host buff
;
filhst:
        ;may have to fill the host buffer
        lda     sekdsk
        sta     hstdsk
        lhld    sektrk
        shld    hsttrk
        lda     sekhst
        sta     hstsec
        lda     rsflag          ;need to read?
        ora     a
        cnz     readhst         ;yes, if 1
        xra     a               ;0 to accum
        sta     hstwrt          ;no pending write
;
match:
        ;copy data to or from buffer
        lda     seksec          ;mask buffer number
        ani     secmsk          ;least signif bits
        mov     l,a             ;ready to shift
        mvi     h,0             ;double count
        rept    7               ;shift left 7
        dad     h
        endm
;       hl has relative host buffer address
        lxi     d,hstbuf
        dad     d               ;hl = host address
        xchg                    ;now in DE
        lhld    dmaadr          ;get/put CP/M data
        mvi     c,128           ;length of move
```

```
        lda     readop          ;which way?
        ora     a
        jnz     rwmove          ;skip if read
;
;       write operation, mark and switch direction
        mvi     a,1
        sta     hstwrt          ;hstwrt = 1
        xchg                    ;source/dest swap
;
rwmove:
        ;C initially 128, DE is source, HL is dest
        ldax    d               ;source character
        inx     d
        mov     m,a             ;to dest
        inx     h
        dcr     c               ;loop 128 times
        jnz     rwmove
;
;       data has been moved to/from host buffer
        lda     wrtype          ;write type
        cpi     wrdir           ;to directory?
        lda     erflag          ;in case of errors
        rnz                     ;no further processing
;
;       clear host buffer for directory write
        ora     a               ;errors?
        rnz                     ;skip if so
        xra     a               ;0 to accum
        sta     hstwrt          ;buffer written
        call    writehst
        lda     erflag
        ret
;
;*****************************************************
;*                                                   *
;*      Utility subroutine for 16-bit compare        *
;*                                                   *
;*****************************************************
sektrkcmp:
        ;HL = .unatrk or .hsttrk, compare with sektrk
        xchg
        lxi     h,sektrk
        ldax    d               ;low byte compare
        cmp     m               ;same?
        rnz                     ;return if not
;       low bytes equal, test high 1s
        inx     d
        inx     h
        ldax    d
        cmp     m       ;sets flags
        ret
;
```

```
;*****************************************************
;*                                                   *
;*      WRITEHST performs the physical write to      *
;*      the host disk, READHST reads the physical    *
;*      disk.                                        *
;*                                                   *
;*****************************************************
writehst:
        ;hstdsk = host disk #, hsttrk = host track #,
        ;hstsec = host sect #. write "hstsiz" bytes
        ;from hstbuf and return error flag in erflag.
        ;return erflag non-zero if error
        ret
;
readhst:
        ;hstdsk = host disk #, hsttrk = host track #,
        ;hstsec = host sect #. read "hstsiz" bytes
        ;into hstbuf and return error flag in erflag.
        ret
;
;*****************************************************
;*                                                   *
;*      Unitialized RAM data areas                   *
;*                                                   *
;*****************************************************
;
sekdsk: ds      1               ;seek disk number
sektrk: ds      2               ;seek track number
seksec: ds      1               ;seek sector number
;
hstdsk: ds      1               ;host disk number
hsttrk: ds      2               ;host track number
hstsec: ds      1               ;host sector number
;
sekhst: ds      1               ;seek shr secshf
hstact: ds      1               ;host active flag
hstwrt: ds      1               ;host written flag
;
unacnt: ds      1               ;unalloc rec cnt
unadsk: ds      1               ;last unalloc disk
unatrk: ds      2               ;last unalloc track
unasec: ds      1               ;last unalloc sector
;
erflag: ds      1               ;error reporting
rsflag: ds      1               ;read sector flag
readop: ds      1               ;1 if read operation
wrtype: ds      1               ;write operation type
dmaadr: ds      2               ;last dma address
hstbuf: ds      hstsiz          ;host buffer
;
```

```
;****************************************************
;*                                                  *
;*      The ENDEF macro invocation goes here        *
;*                                                  *
;****************************************************
        end
```

DIGITAL RESEARCH®

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

CP/M 2.2 INTERFACE GUIDE

Disclaimer

CP/M 2.2 INTERFACE GUIDE

1. INTRODUCTION.

This manual describes CP/M, release 2, system organization including the structure of memory and system entry points. The intention 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.

CP/M is logically divided into four parts, called the Basic I/O System (BIOS), the Basic Disk Operating System (BDOS), the Console command processor (CCP), and the Transient Program Area (TPA). The BIOS is a hardware-dependent module which defines the exact low level interface to a particular computer system which is necessary for peripheral device I/O. Although a standard BIOS is supplied by Digital Research, explicit instructions are provided for field reconfiguration of the BIOS to match nearly any hardware environment (see the Digital Research manual entitled "CP/M Alteration Guide"). The BIOS and BDOS are logically combined into a single module 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 backup storage device. 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 and user programs are executed. The lower portion of memory is reserved for system information and is detailed later sections. Memory organization of the CP/M system in shown below:

```
          ---------------------------
   high  |                           |
 memory  |                           |
         |    FDOS (BDOS+BIOS)       |
 FBASE:  |                           |
          ---------------------------
         |                           |
         |          CCP              |
 CBASE:  |                           |
          ---------------------------
         |                           |
         |                           |
         |                           |
         |          TPA              |
         |                           |
 TBASE:  |                           |
          ---------------------------
         |    system  parameters     |
  BOOT:  |                           |
          ---------------------------
```

The exact memory addresses corresponding to BOOT, TBASE, CBASE, and FBASE vary from version to version, and are described fully in the "CP/M Alteration Guide." All standard CP/M versions, however, assume BOOT = 0000H, which is the base of random access memory. The machine code found at location BOOT performs a system "warm start" which loads and initializes the programs and variables necessary to return control to the CCP. Thus, transient programs need only jump to location BOOT to return control to CP/M at the command level. Further, the standard versions assume TBASE = BOOT+0100H which is normally location 0100H. The principal entry point to the FDOS is at location BOOT+0005H (normally 0005H) where a jump to FBASE is found. The address field at BOOT+0006H (normally 0006H) contains the value of FBASE and can be used to determine the size of available memory, assuming the CCP is being overlayed by a transient program.

Transient programs are loaded into the TPA and executed as follows. The operator communicates with the CCP by typing command lines following each prompt. Each command line takes one of the forms:

```
command
command file1
command file1 file2
```

where "command" is either a built-in function such as 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. The CCP loads the COM file from the disk into memory starting at TBASE and possibly extending up to CBASE.

If the command is followed by one or two file specifications, the CCP prepares one or two file control block (FCB) names in the system parameter area. These optional FCB's are in the form necessary to access files through the FDOS, and are described in the next section.

The transient program receives control from the CCP and begins execution, perhaps using the I/O facilities of the FDOS. The transient program is "called" from the CCP, and thus can simply return to the CCP upon completion of its processing, or can jump to BOOT to pass control back to CP/M. In the first case, the transient program must not use memory above CBASE, while in the latter case, memory up through FBASE-1 is free.

The transient program may use the CP/M I/O facilities to communicate with the operator's console and peripheral devices, including the disk subsystem. The I/O system is accessed by passing a "function number" and an "information address" to CP/M through the FDOS entry point at BOOT+0005H. 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 to the CP/M FDOS. The FDOS, in turn, performs the operation and returns with either a disk read completion indication or an error number indicating that the disk read was unsuccessful. The function numbers and error indicators are given in below.

2. OPERATING SYSTEM CALL CONVENTIONS.

The purpose of this section is to provide detailed information for performing direct operating system calls from user programs. Many of the functions listed below, however, are more simply accessed through the I/O macro library provided with the MAC macro assembler, and listed in the Digital Research manual entitled "MAC Macro Assembler: Language Manual and Applications Guide."

CP/M facilities which are available for access by transient programs fall into two general categories: simple device I/O, and disk file I/O. The simple device operations include:

- Read a Console Character
- Write a Console Character
- Read a Sequential Tape Character
- Write a Sequential Tape Character
- Write a List Device Character
- Get or Set I/O Status
- Print Console Buffer
- Read Console Buffer
- Interrogate Console Ready

The FDOS operations which perform disk Input/Output are

- Disk System Reset
- Drive Selection
- File Creation
- File Open
- File Close
- Directory Search
- File Delete
- File Rename
- Random or Sequential Read
- Random or Sequential Write
- Interrogate Available Disks
- Interrogate Selected Disk
- Set DMA Address
- Set/Reset File Indicators

As mentioned above, access to the FDOS functions is accomplished by passing a function number and information address through the primary entry point at location BOOT+ØØØ5H. In general, the function number is passed in register C with the information address in the double byte pair DE. Single byte values are returned in register A, with double byte values returned in HL (a zero value is returned when the function number is out of range). For reasons of compatibility, register A = L and register B = H upon return in all cases. Note that the register passing conventions of CP/M agree with those of Intel's PL/M systems programming language. The list of CP/M function numbers is given below.

0	System Reset	19	Delete File
1	Console Input	20	Read Sequential
2	Console Output	21	Write Sequential
3	Reader Input	22	Make File
4	Punch Output	23	Rename File
5	List Output	24	Return Login Vector
6	Direct Console I/O	25	Return Current Disk
7	Get I/O Byte	26	Set DMA Address
8	Set I/O Byte	27	Get Addr(Alloc)
9	Print String	28	Write Protect Disk
10	Read Console Buffer	29	Get R/O Vector
11	Get Console Status	30	Set File Attributes
12	Return Version Number	31	Get Addr(Disk Parms)
13	Reset Disk System	32	Set/Get User Code
14	Select Disk	33	Read Random
15	Open File	34	Write Random
16	Close File	35	Compute File Size
17	Search for First	36	Set Random Record
18	Search for Next		

(Functions 28 and 32 should be avoided in application programs to maintain upward compatibility with MP/M.)

Upon entry to a transient program, the CCP leaves the stack pointer set to an eight level stack area with the CCP return address pushed onto the stack, leaving seven levels before overflow occurs. Although this stack is usually not used by a transient program (i.e., most transients return to the CCP though a jump to location 0000H), it is sufficiently large to make CP/M system calls since the FDOS switches to a local stack at system entry. The following assembly language program segment, for example, reads characters continuously until an asterisk is encountered, at which time control returns to the CCP (assuming a standard CP/M system with BOOT = 0000H):

```
BDOS    EQU     0005H       ;STANDARD CP/M ENTRY
CONIN   EQU     1           ;CONSOLE INPUT FUNCTION
;
        ORG     0100H       ;BASE OF TPA
NEXTC:  MVI     C,CONIN     ;READ NEXT CHARACTER
        CALL    BDOS        ;RETURN CHARACTER IN <A>
        CPI     '*'         ;END OF PROCESSING?
        JNZ     NEXTC       ;LOOP IF NOT
        RET                 ;RETURN TO CCP
        END
```

CP/M implements a named file structure on each disk, providing a logical organization which allows any particular file to contain any number of records from completely empty, to the full capacity of the drive. Each drive is logically distinct with a disk directory and file data area. The disk file names are in three parts: the drive select code, the file name consisting of one to eight non-blank characters, and the file type consisting of zero to three non-blank characters. The file type names the generic category of a particular file, while the file name distinguishes individual files in each category. The file types listed below name a few generic categories which have been established, although they are generally arbitrary:

ASM	Assembler Source	PLI	PL/I Source File
PRN	Printer Listing	REL	Relocatable Module
HEX	Hex Machine Code	TEX	TEX Formatter Source
BAS	Basic Source File	BAK	ED Source Backup
INT	Intermediate Code	SYM	SID Symbol File
COM	CCP Command File	$$$	Temporary File

Source files are treated as a sequence of ASCII characters, where each "line" of the source file is followed by a carriage-return line-feed sequence (ØDH followed by ØAH). Thus one 128 byte CP/M record could contain several lines of source text. The end of an ASCII file is denoted by a control-Z character (1AH) or a real end of file, returned by the CP/M read operation. Control-Z characters embedded within machine code files (e.g., COM files) are ignored, however, and the end of file condition returned by CP/M is used to terminate read operations.

Files in CP/M can be thought of as a sequence of up to 65536 records of 128 bytes each, numbered from Ø through 65535, thus allowing a maximum of 8 megabytes per file. Note, however, that although the records may be considered logically contiguous, they may not be physically contiguous in the disk data area. Internally, all files are broken into 16K byte segments called logical extents, so that counters are easily maintained as 8-bit values. Although the decomposition into extents is discussed in the paragraphs which follow, they are of no particular consequence to the programmer since each extent is automatically accessed in both sequential and random access modes.

In the file operations starting with function number 15, DE usually addresses a file control block (FCB). Transient programs often use the default file control block area reserved by CP/M at location BOOT+ØØ5CH (normally ØØ5CH) for simple file operations. The basic unit of file information is a 128 byte record used for all file operations, thus a default location for disk I/O is provided by CP/M at location BOOT+ØØ8ØH (normally ØØ8ØH) which is the initial default DMA address (see function 26). All directory operations take place in a reserved area which does not affect write buffers as was the case in release 1, with the exception of Search First and Search Next, where compatibility is required.

The File Control Block (FCB) data area consists of a sequence of 33 bytes for sequential access and a series of 36 bytes in the case that the file is accessed randomly. The default file control block normally located at ØØ5CH can be used for random access files, since the three bytes starting at BOOT+ØØ7DH are available for this purpose. The FCB format is shown with the following fields:

```
-------------------------------------------------------------
|dr|f1|f2|/ /|f8|t1|t2|t3|ex|s1|s2|rc|d0|/ /|dn|cr|r0|r1|r2|
-------------------------------------------------------------
 00 01 02 ... 08 09 10 11 12 13 14 15 16 ... 31 32 33 34 35
```

where

```
   dr        drive code (0 - 16)
             0 => use default drive for file
             1 => auto disk select drive A,
             2 => auto disk select drive B,
             ...
             16=> auto disk select drive P.

   f1...f8   contain the file name in ASCII
             upper case, with high bit = 0

   t1,t2,t3  contain the file type in ASCII
             upper case, with high bit = 0
             t1', t2', and t3' denote the
             bit of these positions,
             t1' = 1 => Read/Only file,
             t2' = 1 => SYS file, no DIR list

    ex       contains the current extent number,
             normally set to 00 by the user, but
             in range 0 - 31 during file I/O

    s1       reserved for internal system use

    s2       reserved for internal system use, set
             to zero on call to OPEN, MAKE, SEARCH

    rc       record count for extent "ex,"
             takes on values from 0 - 128

    d0...dn  filled-in by CP/M, reserved for
             system use

    cr       current record to read or write in
             a sequential file operation, normally
             set to zero by user

    r0,r1,r2 optional random record number in the
             range 0-65535, with overflow to r2,
             r0,r1 constitute a 16-bit value with
             low byte r0, and high byte r1
```

Each file being accessed through CP/M must have a corresponding FCB which provides the name and allocation information for all subsequent file operations. When accessing files, it is the programmer's responsibility to fill the lower sixteen bytes of the FCB and initialize the "cr" field. Normally, bytes 1 through 11 are set to the ASCII character values for the file name and file type, while all other fields are zero.

FCB's are stored in a directory area of the disk, and are brought into central memory before proceeding with file operations (see the OPEN and MAKE functions). The memory copy of the FCB is updated as file operations take place and later recorded permanently on disk at the termination of the file operation (see the CLOSE command).

The CCP constructs the first sixteen bytes of two optional FCB's for a transient by scanning the remainder of the line following the transient name, denoted by "file1" and "file2" in the prototype command line described above, with unspecified fields set to ASCII blanks. The first FCB is constructed at location BOOT+005CH, and can be used as-is for subsequent file operations. The second FCB occupies the d0 ... dn portion of the first FCB, and must be moved to another area of memory before use. If, for example, the operator types

PROGNAME B:X.ZOT Y.ZAP

the file PROGNAME.COM is loaded into the TPA, and the default FCB at BOOT+005CH is initialized to drive code 2, file name "X" and file type "ZOT". The second drive code takes the default value 0, which is placed at BOOT+006CH, with the file name "Y" placed into location BOOT+006DH and file type "ZAP" located 8 bytes later at BOOT+0075H. All remaining fields through "cr" are set to zero. Note again that it is the programmer's responsibility to move this second file name and type to another area, usually a separate file control block, before opening the file which begins at BOOT+005CH, due to the fact that the open operation will overwrite the second name and type.

If no file names are specified in the original command, then the fields beginning at BOOT+005DH and BOOT+006DH contain blanks. In all cases, the CCP translates lower case alphabetics to upper case to be consistent with the CP/M file naming conventions.

As an added convenience, the default buffer area at location BOOT+0080H is initialized to the command line tail typed by the operator following the program name. The first position contains the number of characters, with the characters themselves following the character count. Given the above command line, the area beginning at BOOT+0080H is initialized as follows:

```
BOOT+0080H:
+00 +01 +02 +03 +04 +05 +06 +07 +08 +09 +10 +11 +12 +13 +14
 14 " "  "B" ":" "X" "." "Z" "O" "T" " "  "Y" "." "Z" "A" "P"
```

where the characters are translated to upper case ASCII with uninitialized memory following the last valid character. Again, it is the responsibility of the programmer to extract the information from this buffer before any file operations are performed, unless the default DMA address is explicitly changed.

The individual functions are described in detail in the pages which follow.

```
***************************************
*                                     *
*  FUNCTION Ø:  System Reset          *
*                                     *
***************************************
*  Entry Parameters:                  *
*      Register   C:  ØØH             *
***************************************
```

The system reset function returns control to the CP/M operating system at the CCP level. The CCP re-initializes the disk subsystem by selecting and logging-in disk drive A. This function has exactly the same effect as a jump to location BOOT.

```
***************************************
*                                     *
*  FUNCTION 1:  CONSOLE INPUT         *
*                                     *
***************************************
*  Entry Parameters:                  *
*      Register   C:  Ø1H             *
*                                     *
*  Returned   Value:                  *
*      Register   A:  ASCII Character *
***************************************
```

The console input function reads the next console character to register A. Graphic characters, along with carriage return, line feed, and backspace (ctl-H) are echoed to the console. Tab characters (ctl-I) are expanded in columns of eight characters. A check is made for start/stop scroll (ctl-S) and start/stop printer echo (ctl-P). The FDOS does not return to the calling program until a character has been typed, thus suspending execution if a character is not ready.

```
***************************************
*                                     *
*  FUNCTION 2:  CONSOLE OUTPUT        *
*                                     *
***************************************
*  Entry Parameters:                  *
*      Register   C:  Ø2H             *
*      Register   E:  ASCII Character *
*                                     *
***************************************
```

The ASCII character from register E is sent to the console device. Similar to function 1, tabs are expanded and checks are made for start/stop scroll and printer echo.

```
***************************************
*                                     *
*  FUNCTION 3:  READER INPUT          *
*                                     *
***************************************
*  Entry Parameters:                  *
*      Register   C:  03H             *
*                                     *
*  Returned   Value:                  *
*      Register   A:  ASCII Character *
***************************************
```

The Reader Input function reads the next character from the logical reader into register A (see the IOBYTE definition in the "CP/M Alteration Guide"). Control does not return until the character has been read.

```
***************************************
*                                     *
*  FUNCTION 4:  PUNCH OUTPUT          *
*                                     *
***************************************
*  Entry Parameters:                  *
*      Register   C:  04H             *
*      Register   E:  ASCII Character *
*                                     *
***************************************
```

The Punch Output function sends the character from register E to the logical punch device.

```
***************************************
*                                     *
*  FUNCTION 5:  LIST OUTPUT           *
*                                     *
***************************************
*  Entry Parameters:                  *
*      Register   C:  05H             *
*      Register   E:  ASCII Character *
*                                     *
***************************************
```

The List Output function sends the ASCII character in register E to the logical listing device.

```
****************************************
*                                      *
*  FUNCTION 6:  DIRECT CONSOLE I/O     *
*                                      *
****************************************
*  Entry Parameters:                   *
*      Register   C:  Ø6H              *
*      Register   E:  ØFFH (input) or  *
*                     char (output)    *
*                                      *
*  Returned   Value:                   *
*      Register   A:  char or status   *
*                     (no value)       *
****************************************
```

Direct console I/O is supported under CP/M for those specialized applications where unadorned console input and output is required. Use of this function should, in general, be avoided since it bypasses all of CP/M's normal control character functions (e.g., control-S and control-P). Programs which perform direct I/O through the BIOS under previous releases of CP/M, however, should be changed to use direct I/O under BDOS so that they can be fully supported under future releases of MP/M and CP/M.

Upon entry to function 6, register E either contains hexadecimal FF, denoting a console input request, or register E contains an ASCII character. If the input value is FF, then function 6 returns A = ØØ if no character is ready, otherwise A contains the next console input character.

If the input value in E is not FF, then function 6 assumes that E contains a valid ASCII character which is sent to the console.

```
***************************************
*                                     *
*   FUNCTION 7:  GET I/O BYTE         *
*                                     *
***************************************
*  Entry Parameters:                  *
*       Register   C:  Ø7H            *
*                                     *
*  Returned   Value:                  *
*       Register   A:  I/O Byte Value *
***************************************
```

The Get I/O Byte function returns the current value of IOBYTE in register A. See the "CP/M Alteration Guide" for IOBYTE definition.

```
***************************************
*                                     *
*   FUNCTION 8:  SET I/O BYTE         *
*                                     *
***************************************
*  Entry Parameters:                  *
*       Register   C:  Ø8H            *
*       Register   E:  I/O Byte Value *
*                                     *
***************************************
```

The Set I/O Byte function changes the system IOBYTE value to that given in register E.

```
***************************************
*                                     *
*   FUNCTION 9:  PRINT STRING         *
*                                     *
***************************************
*  Entry Parameters:                  *
*       Register   C:  Ø9H            *
*       Registers DE:  String Address *
*                                     *
***************************************
```

The Print String function sends the character string stored in memory at the location given by DE to the console device, until a "$" is encountered in the string. Tabs are expanded as in function 2, and checks are made for start/stop scroll and printer echo.

```
****************************************
*                                      *
*  FUNCTION 10: READ CONSOLE BUFFER    *
*                                      *
****************************************
*  Entry Parameters:                   *
*      Register   C:  0AH              *
*      Registers DE:  Buffer Address   *
*                                      *
*  Returned   Value:                   *
*      Console Characters in Buffer    *
****************************************
```

The Read Buffer function reads a line of edited console input into a buffer addressed by registers DE. Console input is terminated when either the input buffer overflows. The Read Buffer takes the form:

```
DE: +0 +1 +2 +3 +4 +5 +6 +7 +8      . . .     +n
   -------------------------------------------------
   |mx|nc|c1|c2|c3|c4|c5|c6|c7|      . . .     |??|
   -------------------------------------------------
```

where "mx" is the maximum number of characters which the buffer will hold (1 to 255), "nc" is the number of characters read (set by FDOS upon return), followed by the characters read from the console. if nc < mx, then uninitialized positions follow the last character, denoted by "??" in the above figure. A number of control functions are recognized during line editing:

```
rub/del removes and echoes the last character
 ctl-C  reboots when at the beginning of line
 ctl-E  causes physical end of line
 ctl-H  backspaces one character position
 ctl-J  (line feed) terminates input line
 ctl-M  (return) terminates input line
 ctl-R  retypes the current line after new line
 ctl-U  removes currnt line after new line
 ctl-X  backspaces to beginning of current line
```

Note also that certain functions which return the carriage to the leftmost position (e.g., ctl-X) do so only to the column position where the prompt ended (in earlier releases, the carriage returned to the extreme left margin). This convention makes operator data input and line correction more legible.

```
***************************************
*                                     *
*  FUNCTION 11: GET CONSOLE STATUS    *
*                                     *
***************************************
*  Entry Parameters:                  *
*      Register   C:  ØBH             *
*                                     *
*  Returned   Value:                  *
*      Register   A:  Console Status  *
***************************************
```

The Console Status function checks to see if a character has been typed at the console. If a character is ready, the value ØFFH is returned in register A. Otherwise a ØØH value is returned.

```
***************************************
*                                     *
*  FUNCTION 12: RETURN VERSION NUMBER *
*                                     *
***************************************
*  Entry Parameters:                  *
*      Register   C:  ØCH             *
*                                     *
*  Returned   Value:                  *
*      Registers HL:  Version Number  *
***************************************
```

Function 12 provides information which allows version independent programming. A two-byte value is returned, with H = ØØ designating the CP/M release (H = Ø1 for MP/M), and L = ØØ for all releases previous to 2.Ø. CP/M 2.Ø returns a hexadecimal 2Ø in register L, with subsequent version 2 releases in the hexadecimal range 21, 22, through 2F. Using function 12, for example, you can write application programs which provide both sequential and random access functions, with random access disabled when operating under early releases of CP/M.

```
*************************************
*                                   *
*  FUNCTION 13: RESET DISK SYSTEM   *
*                                   *
*************************************
*  Entry Parameters:                *
*      Register   C:   0DH          *
*                                   *
*************************************
```

The Reset Disk Function is used to programmatically restore the file system to a reset state where all disks are set to read/write (see functions 28 and 29), only disk drive A is selected, and the default DMA address is reset to BOOT+0080H. This function can be used, for example, by an application program which requires a disk change without a system reboot.

```
*************************************
*                                   *
*  FUNCTION 14: SELECT DISK         *
*                                   *
*************************************
*  Entry Parameters:                *
*      Register   C:   0EH          *
*      Register   E:   Selected Disk *
*                                   *
*************************************
```

The Select Disk function designates the disk drive named in register E as the default disk for subsequent file operations, with E = 0 for drive A, 1 for drive B, and so-forth through 15 corresponding to drive P in a full sixteen drive system. The drive is placed in an "on-line" status which, in particular, activates its directory until the next cold start, warm start, or disk system reset operation. If the disk media is changed while it is on-line, the drive automatically goes to a read/only status in a standard CP/M environment (see function 28). FCB's which specify drive code zero (dr = 00H) automatically reference the currently selected default drive. Drive code values between 1 and 16, however, ignore the selected default drive and directly reference drives A through P.

```
***************************************
*                                     *
*  FUNCTION 15: OPEN FILE             *
*                                     *
***************************************
*  Entry Parameters:                  *
*      Register   C:  ØFH             *
*      Registers DE:  FCB Address     *
*                                     *
*  Returned   Value:                  *
*      Register   A:  Directory Code  *
***************************************
```

The Open File operation is used to activate a file which currently exists in the disk directory for the currently active user number. The FDOS scans the referenced disk directory for a match in positions 1 through 14 of the FCB referenced by DE (byte s1 is automatically zeroed), where an ASCII question mark (3FH) matches any directory character in any of these positions. Normally, no question marks are included and, further, bytes "ex" and "s2" of the FCB are zero.

If a directory element is matched, the relevant directory information is copied into bytes dØ through dn of the FCB, thus allowing access to the files through subsequent read and write operations. Note that an existing file must not be accessed until a sucessful open operation is completed. Upon return, the open function returns a "directory code" with the value Ø through 3 if the open was successful, or ØFFH (255 decimal) if the file cannot be found. If question marks occur in the FCB then the first matching FCB is activated. Note that the current record ("cr") must be zeroed by the program if the file is to be accessed sequentially from the first record.

```
***************************************
*                                     *
*  FUNCTION 16: CLOSE FILE            *
*                                     *
***************************************
*  Entry Parameters:                  *
*      Register    C:  10H            *
*      Registers DE:  FCB Address     *
*                                     *
*  Returned   Value:                  *
*      Register    A:  Directory Code *
***************************************
```

The Close File function performs the inverse of the open file function. Given that the FCB addressed by DE has been previously activated through an open or make function (see functions 15 and 22), the close function permanently records the new FCB in the referenced disk directory. The FCB matching process for the close is identical to the open function. The directory code returned for a successful close operation is 0, 1, 2, or 3, while a 0FFH (255 decimal) is returned if the file name cannot be found in the directory. A file need not be closed if only read operations have taken place. If write operations have occurred, however, the close operation is necessary to permanently record the new directory information.

```
*****************************************
*                                       *
*  FUNCTION 17: SEARCH FOR FIRST        *
*                                       *
*****************************************
*  Entry Parameters:                    *
*      Register   C:  11H               *
*      Registers DE:  FCB Address       *
*                                       *
*  Returned   Value:                    *
*      Register   A:  Directory Code    *
*****************************************
```

Search First scans the directory for a match with the file given by the FCB addressed by DE. The value 255 (hexadecimal FF) is returned if the file is not found, otherwise Ø, 1, 2, or 3 is returned indicating the file is present. In the case that the file is found, the current DMA address is filled with the record containing the directory entry, and the relative starting position is A * 32 (i.e., rotate the A register left 5 bits, or ADD A five times). Although not normally required for application programs, the directory information can be extracted from the buffer at this position.

An ASCII question mark (63 decimal, 3F hexadecimal) in any position from "f1" through "ex" matches the corresponding field of any directory entry on the default or auto-selected disk drive. If the "dr" field contains an ASCII question mark, then the auto disk select function is disabled, the default disk is searched, with the search function returning any matched entry, allocated or free, belonging to any user number. This latter function is not normally used by application programs, but does allow complete flexibility to scan all current directory values. If the "dr" field is not a question mark, the "s2" byte is automatically zeroed.

```
*****************************************
*                                       *
*  FUNCTION 18: SEARCH FOR NEXT         *
*                                       *
*****************************************
*  Entry Parameters:                    *
*      Register   C:  12H               *
*                                       *
*  Returned   Value:                    *
*      Register   A:  Directory Code    *
*****************************************
```

The Search Next function is similar to the Search First function, except that the directory scan continues from the last matched entry. Similar to function 17, function 18 returns the decimal value 255 in A when no more directory items match.

```
****************************************
*                                      *
*  FUNCTION 19: DELETE FILE            *
*                                      *
****************************************
*  Entry Parameters:                   *
*      Register   C:  13H              *
*      Registers DE:  FCB Address      *
*                                      *
*  Returned   Value:                   *
*      Register   A:  Directory Code   *
****************************************
```

The Delete File function removes files which match the FCB addressed by DE. The filename and type may contain ambiguous references (i.e., question marks in various positions), but the drive select code cannot be ambiguous, as in the Search and Search Next functions.

Function 19 returns a decimal 255 if the referenced file or files cannot be found, otherwise a value in the range Ø to 3 is returned.

```
****************************************
*                                      *
*  FUNCTION 2Ø: READ SEQUENTIAL        *
*                                      *
****************************************
*  Entry Parameters:                   *
*      Register   C:  14H              *
*      Registers DE:  FCB Address      *
*                                      *
*  Returned   Value:                   *
*      Register   A:  Directory Code   *
****************************************
```

Given that the FCB addressed by DE has been activated through an open or make function (numbers 15 and 22), the Read Sequential function reads the next 128 byte record from the file into memory at the current DMA address. the record is read from position "cr" of the extent, and the "cr" field is automatically incremented to the next record position. If the "cr" field overflows then the next logical extent is automatically opened and the "cr" field is reset to zero in preparation for the next read operation. The value ØØH is returned in the A register if the read operation was successful, while a non-zero value is returned if no data exists at the next record position (e.g., end of file occurs).

```
****************************************
*                                      *
*   FUNCTION 21: WRITE SEQUENTIAL      *
*                                      *
****************************************
*  Entry Parameters:                   *
*       Register    C:   15H           *
*       Registers DE:   FCB Address    *
*                                      *
*  Returned   Value:                   *
*       Register    A:  Directory Code *
****************************************
```

Given that the FCb addressed by DE has been activated through an open or make function (numbers 15 and 22), the Write Sequential function writes the 128 byte data record at the current DMA address to the file named by the FCB. the record is placed at position "cr" of the file, and the "cr" field is automatically incremented to the next record position. If the "cr" field overflows then the next logical extent is automatically opened and the "cr" field is reset to zero in preparation for the next write operation. Write operations can take place into an existing file, in which case newly written records overlay those which already exist in the file. Register A = ØØH upon return from a successful write operation, while a non-zero value indicates an unsuccessful write due to a full disk.

```
****************************************
*                                      *
*   FUNCTION 22: MAKE FILE             *
*                                      *
****************************************
*  Entry Parameters:                   *
*       Register    C:   16H           *
*       Registers DE:   FCB Address    *
*                                      *
*  Returned   Value:                   *
*       Register    A:  Directory Code *
****************************************
```

The Make File operation is similar to the open file operation except that the FCB must name a file which does not exist in the currently referenced disk directory (i.e., the one named explicitly by a non-zero "dr" code, or the default disk if "dr" is zero). The FDOS creates the file and initializes both the directory and main memory value to an empty file. The programmer must ensure that no duplicate file names occur, and a preceding delete operation is sufficient if there is any possibility of duplication. Upon return, register A = Ø, 1, 2, or 3 if the operation was successful and ØFFH (255 decimal) if no more directory space is available. The make function has the side-effect of activating the FCB and thus a subsequent open is not necessary.

```
****************************************
*                                      *
*   FUNCTION 23: RENAME FILE           *
*                                      *
****************************************
*   Entry Parameters:                  *
*       Register   C:  17H             *
*       Registers DE:  FCB Address     *
*                                      *
*   Returned    Value:                 *
*       Register   A:  Directory Code  *
****************************************
```

The Rename function uses the FCB addressed by DE to change all occurrences of the file named in the first 16 bytes to the file named in the second 16 bytes. The drive code "dr" at position 0 is used to select the drive, while the drive code for the new file name at position 16 of the FCB is assumed to be zero. Upon return, register A is set to a value between 0 and 3 if the rename was successful, and 0FFH (255 decimal) if the first file name could not be found in the directory scan.

```
****************************************
*                                      *
*   FUNCTION 24: RETURN LOGIN VECTOR   *
*                                      *
****************************************
*   Entry Parameters:                  *
*       Register   C:  18H             *
*                                      *
*   Returned    Value:                 *
*       Registers HL:  Login Vector    *
****************************************
```

The login vector value returned by CP/M is a 16-bit value in HL, where the least significant bit of L corresponds to the first drive A, and the high order bit of H corresponds to the sixteenth drive, labelled P. A "0" bit indicates that the drive is not on-line, while a "1" bit marks an drive that is actively on-line due to an explicit disk drive selection, or an implicit drive select caused by a file operation which specified a non-zero "dr" field. Note that compatibility is maintained with earlier releases, since registers A and L contain the same values upon return.

```
***************************************
*                                     *
*  FUNCTION 25: RETURN CURRENT DISK   *
*                                     *
***************************************
*  Entry Parameters:                  *
*      Register   C:  19H             *
*                                     *
*  Returned   Value:                  *
*      Register   A:  Current Disk    *
***************************************
```

Function 25 returns the currently selected default disk number in register A. The disk numbers range from Ø through 15 corresponding to drives A through P.

```
***************************************
*                                     *
*  FUNCTION 26: SET DMA ADDRESS       *
*                                     *
***************************************
*  Entry Parameters:                  *
*      Register   C:  1AH             *
*      Registers DE:  DMA Address     *
*                                     *
***************************************
```

"DMA" is an acronym for Direct Memory Address, which is often used in connection with disk controllers which directly access the memory of the mainframe computer to transfer data to and from the disk subsystem. Although many computer systems use non-DMA access (i.e., the data is transfered through programmed I/O operations), the DMA address has, in CP/M, come to mean the address at which the 128 byte data record resides before a disk write and after a disk read. Upon cold start, warm start, or disk system reset, the DMA address is automatically set to BOOT+ØØ8ØH. The Set DMA function, however, can be used to change this default value to address another area of memory where the data records reside. Thus, the DMA address becomes the value specified by DE until it is changed by a subsequent Set DMA function, cold start, warm start, or disk system reset.

```
****************************************
*                                      *
*  FUNCTION 27: GET ADDR(ALLOC)        *
*                                      *
****************************************
*  Entry Parameters:                   *
*      Register   C:  1BH              *
*                                      *
*  Returned   Value:                   *
*      Registers HL:  ALLOC Address    *
****************************************
```

An "allocation vector" is maintained in main memory for each on-line disk drive. Various system programs use the information provided by the allocation vector to determine the amount of remaining storage (see the STAT program). Function 27 returns the base address of the allocation vector for the currently selected disk drive. The allocation information may, however, be invalid if the selected disk has been marked read/only. Although this function is not normally used by application programs, additional details of the allocation vector are found in the "CP/M Alteration Guide."

```
****************************************
*                                      *
*  FUNCTION 28: WRITE PROTECT DISK     *
*                                      *
****************************************
*  Entry Parameters:                   *
*      Register   C:  1CH              *
*                                      *
****************************************
```

The disk write protect function provides temporary write protection for the currently selected disk. Any attempt to write to the disk, before the next cold or warm start operation produces the message

Bdos Err on d: R/O

```
***************************************
*                                     *
*  FUNCTION 29: GET READ/ONLY VECTOR  *
*                                     *
***************************************
*  Entry Parameters:                  *
*      Register   C:  1DH             *
*                                     *
*  Returned   Value:                  *
*      Registers HL:  R/O Vector Value*
***************************************
```

Function 29 returns a bit vector in register pair HL which indicates drives which have the temporary read/only bit set. Similar to function 24, the least significant bit corresponds to drive A, while the most significant bit corresponds to drive P. The R/O bit is set either by an explicit call to function 28, or by the automatic software mechanisms within CP/M which detect changed disks.

```
***************************************
*                                     *
*  FUNCTION 30: SET FILE ATTRIBUTES   *
*                                     *
***************************************
*  Entry Parameters:                  *
*      Register   C:  1EH             *
*      Registers DE:  FCB Address     *
*                                     *
*  Returned   Value:                  *
*      Register   A:  Directory Code  *
***************************************
```

The Set File Attributes function allows programmatic manipulation of permanent indicators attached to files. In particular, the R/O and System attributes (t1' and t2') can be set or reset. The DE pair addresses an unambiguous file name with the appropriate attributes set or reset. Function 30 searches for a match, and changes the matched directory entry to contain the selected indicators. Indicators f1' through f4' are not presently used, but may be useful for applications programs, since they are not involved in the matching process during file open and close operations. Indicators f5' through f8' and t3' are reserved for future system expansion.

```
***************************************
*                                     *
*  FUNCTION 31: GET ADDR(DISK PARMS)  *
*                                     *
***************************************
*  Entry Parameters:                  *
*      Register   C:  1FH             *
*                                     *
*  Returned   Value:                  *
*      Registers HL:  DPB Address     *
***************************************
```

The address of the BIOS resident disk parameter block is returned in HL as a result of this function call. This address can be used for either of two purposes. First, the disk parameter values can be extracted for display and space computation purposes, or transient programs can dynamically change the values of current disk parameters when the disk environment changes, if required. Normally, application programs will not require this facility.

```
***************************************
*                                     *
*  FUNCTION 32: SET/GET USER CODE     *
*                                     *
***************************************
*  Entry Parameters:                  *
*      Register   C:  20H             *
*      Register   E:  ØFFH (get) or   *
*                     User Code (set) *
*                                     *
*  Returned   Value:                  *
*      Register   A:  Current Code or *
*                     (no value)      *
***************************************
```

An application program can change or interrogate the currently active user number by calling function 32. If register E = ØFFH, then the value of the current user number is returned in register A, where the value is in the range Ø to 31. If register E is not ØFFH, then the current user number is changed to the value of E (modulo 32).

```
***************************************
*                                     *
*  FUNCTION 33: READ RANDOM           *
*                                     *
***************************************
*  Entry Parameters:                  *
*      Register   C:  21H             *
*      Registers DE:  FCB Address     *
*                                     *
*  Returned  Value:                   *
*      Register   A:  Return Code     *
***************************************
```

The Read Random function is similar to the sequential file read operation of previous releases, except that the read operation takes place at a particular record number, selected by the 24-bit value constructed from the three byte field following the FCB (byte positions r0 at 33, r1 at 34, and r2 at 35). Note that the sequence of 24 bits is stored with least significant byte first (r0), middle byte next (r1), and high byte last (r2). CP/M does not reference byte r2, except in computing the size of a file (function 35). Byte r2 must be zero, however, since a non-zero value indicates overflow past the end of file.

Thus, the r0,r1 byte pair is treated as a double-byte, or "word" value, which contains the record to read. This value ranges from 0 to 65535, providing access to any particular record of the 8 megabyte file. In order to process a file using random access, the base extent (extent 0) must first be opened. Although the base extent may or may not contain any allocated data, this ensures that the file is properly recorded in the directory, and is visible in DIR requests. The selected record number is then stored into the random record field (r0,r1), and the BDOS is called to read the record. Upon return from the call, register A either contains an error code, as listed below, or the value 00 indicating the operation was successful. In the latter case, the current DMA address contains the randomly accessed record. Note that contrary to the sequential read operation, the record number is not advanced. Thus, subsequent random read operations continue to read the same record.

Upon each random read operation, the logical extent and current record values are automatically set. Thus, the file can be sequentially read or written, starting from the current randomly accessed position. Note, however, that in this case, the last randomly read record will be re-read as you switch from random mode to sequential read, and the last record will be re-written as you switch to a sequential write operation. You can, of course, simply advance the random record position following each random read or write to obtain the effect of a sequential I/O operation.

Error codes returned in register A following a random read are listed below.

Ø1 reading unwritten data
Ø2 (not returned in random mode)
Ø3 cannot close current extent
Ø4 seek to unwritten extent
Ø5 (not returned in read mode)
Ø6 seek past physical end of disk

Error code Ø1 and Ø4 occur when a random read operation accesses a data block which has not been previously written, or an extent which has not been created, which are equivalent conditions. Error 3 does not normally occur under proper system operation, but can be cleared by simply re-reading, or re-opening extent zero as long as the disk is not physically write protected. Error code Ø6 occurs whenever byte r2 is non-zero under the current 2.Ø release. Normally, non-zero return codes can be treated as missing data, with zero return codes indicating operation complete.

```
***************************************
*                                     *
*  FUNCTION 34: WRITE RANDOM          *
*                                     *
***************************************
*  Entry Parameters:                  *
*       Register   C:  22H            *
*       Registers DE:  FCB Address    *
*                                     *
*  Returned  Value:                   *
*       Register   A:  Return Code    *
***************************************
```

The Write Random operation is initiated similar to the Read Random call, except that data is written to the disk from the current DMA address. Further, if the disk extent or data block which is the target of the write has not yet been allocated, the allocation is performed before the write operation continues. As in the Read Random operation, the random record number is not changed as a result of the write. The logical extent number and current record positions of the file control block are set to correspond to the random record which is being written. Again, sequential read or write operations can commence following a random write, with the notation that the currently addressed record is either read or rewritten again as the sequential operation begins. You can also simply advance the random record position following each write to get the effect of a sequential write operation. Note that in particular, reading or writing the last record of an extent in random mode does not cause an automatic extent switch as it does in sequential mode.

The error codes returned by a random write are identical to the random read operation with the addition of error code Ø5, which indicates that a new extent cannot be created due to directory overflow.

```
***************************************
*                                     *
*  FUNCTION 35: COMPUTE FILE SIZE     *
*                                     *
***************************************
*  Entry Parameters:                  *
*      Register   C:  23H             *
*      Registers DE:  FCB Address     *
*                                     *
*  Returned  Value:                   *
*      Random Record Field Set        *
***************************************
```

When computing the size of a file, the DE register pair addresses an FCB in random mode format (bytes r0, r1, and r2 are present). The FCB contains an unambiguous file name which is used in the directory scan. Upon return, the random record bytes contain the "virtual" file size which is, in effect, the record address of the record following the end of the file. if, following a call to function 35, the high record byte r2 is 01, then the file contains the maximum record count 65536. Otherwise, bytes r0 and r1 constitute a 16-bit value (r0 is the least significant byte, as before) which is the file size.

Data can be appended to the end of an existing file by simply calling function 35 to set the random record position to the end of file, then performing a sequence of random writes starting at the preset record address.

The virtual size of a file corresponds to the physical size when the file is written sequentially. If, instead, the file was created in random mode and "holes" exist in the allocation, then the file may in fact contain fewer records than the size indicates. If, for example, only the last record of an eight megabyte file is written in random mode (i.e., record number 65535), then the virtual size is 65536 records, although only one block of data is actually allocated.

```
***************************************
*                                     *
*  FUNCTION 36: SET RANDOM RECORD     *
*                                     *
***************************************
*  Entry Parameters:                  *
*      Register   C:  24H             *
*      Registers DE:  FCB Address     *
*                                     *
*  Returned   Value:                  *
*      Random Record Field Set        *
***************************************
```

The Set Random Record function causes the BDOS to automatically produce the random record position from a file which has been read or written sequentially to a particular point. The function can be useful in two ways.

First, it is often necessary to initially read and scan a sequential file to extract the positions of various "key" fields. As each key is encountered, function 36 is called to compute the random record position for the data corresponding to this key. If the data unit size is 128 bytes, the resulting record position is placed into a table with the key for later retrieval. After scanning the entire file and tabularizing the keys and their record numbers, you can move instantly to a particular keyed record by performing a random read using the corresponding random record number which was saved earlier. The scheme is easily generalized when variable record lengths are involved since the program need only store the buffer-relative byte position along with the key and record number in order to find the exact starting position of the keyed data at a later time.

A second use of function 36 occurs when switching from a sequential read or write over to random read or write. A file is sequentially accessed to a particular point in the file, function 36 is called which sets the record number, and subsequent random read and write operations continue from the selected point in the file.

3. A SAMPLE FILE-TO-FILE COPY PROGRAM.

The program shown below provides a relatively simple example of file operations. The program source file is created as COPY.ASM using the CP/M ED program and then assembled using ASM or MAC, resulting in a "HEX" file. The LOAD program is the used to produce a COPY.COM file which executes directly under the CCP. The program begins by setting the stack pointer to a local area, and then proceeds to move the second name from the default area at 006CH to a 33-byte file control block called DFCB. The DFCB is then prepared for file operations by clearing the current record field. At this point, the source and destination FCB's are ready for processing since the SFCB at 005CH is properly set-up by the CCP upon entry to the COPY program. That is, the first name is placed into the default fcb, with the proper fields zeroed, including the current record field at 007CH. The program continues by opening the source file, deleting any exising destination file, and then creating the destination file. If all this is successful, the program loops at the label COPY until each record has been read from the source file and placed into the destination file. Upon completion of the data transfer, the destination file is closed and the program returns to the CCP command level by jumping to BOOT.

```
                ;       sample file-to-file copy program
                ;
                ;       at the ccp level, the command
                ;
                ;               copy a:x.y b:u.v
                ;
                ;       copies the file named x.y from drive
                ;       a to a file named u.v on drive b.
                ;
0000 =          boot    equ     0000h   ; system reboot
0005 =          bdos    equ     0005h   ; bdos entry point
005c =          fcb1    equ     005ch   ; first file name
005c =          sfcb    equ     fcb1    ; source fcb
006c =          fcb2    equ     006ch   ; second file name
0080 =          dbuff   equ     0080h   ; default buffer
0100 =          tpa     equ     0100h   ; beginning of tpa
                ;
0009 =          printf  equ     9       ; print buffer func#
000f =          openf   equ     15      ; open file func#
0010 =          closef  equ     16      ; close file func#
0013 =          deletef equ     19      ; delete file func#
0014 =          readf   equ     20      ; sequential read
0015 =          writef  equ     21      ; sequential write
0016 =          makef   equ     22      ; make file func#
                ;
0100                    org     tpa     ; beginning of tpa
0100 311b02             lxi     sp,stack; local stack
                ;
                ;       move second file name to dfcb
0103 0e10               mvi     c,16    ; half an fcb
```

```
0105 116c00           lxi     d,fcb2   ; source of move
0108 21da01           lxi     h,dfcb   ; destination fcb
010b 1a      mfcb:    ldax    d        ; source fcb
010c 13               inx     d        ; ready next
010d 77               mov     m,a      ; dest fcb
010e 23               inx     h        ; ready next
010f 0d               dcr     c        ; count 16...0
0110 c20b01           jnz     mfcb     ; loop 16 times
             ;
             ;        name has been moved, zero cr
0113 af               xra     a        ; a = 00h
0114 32fa01           sta     dfcbcr   ; current rec = 0
             ;
             ;        source and destination fcb's ready
             ;
0117 115c00           lxi     d,sfcb   ; source file
011a cd6901           call    open     ; error if 255
011d 118701           lxi     d,nofile; ready message
0120 3c               inr     a        ; 255 becomes 0
0121 cc6101           cz      finis    ; done if no file
             ;
             ;        source file open, prep destination
0124 11da01           lxi     d,dfcb   ; destination
0127 cd7301           call    delete   ; remove if present
             ;
012a 11da01           lxi     d,dfcb   ; destination
012d cd8201           call    make     ; create the file
0130 119601           lxi     d,nodir  ; ready message
0133 3c               inr     a        ; 255 becomes 0
0134 cc6101           cz      finis    ; done if no dir space
             ;
             ;        source file open, dest file open
             ;        copy until end of file on source
             ;
0137 115c00  copy:    lxi     d,sfcb   ; source
013a cd7801           call    read     ; read next record
013d b7               ora     a        ; end of file?
013e c25101           jnz     eofile   ; skip write if so
             ;
             ;        not end of file, write the record
0141 11da01           lxi     d,dfcb   ; destination
0144 cd7d01           call    write    ; write record
0147 11a901           lxi     d,space  ; ready message
014a b7               ora     a        ; 00 if write ok
014b c46101           cnz     finis    ; end if so
014e c33701           jmp     copy     ; loop until eof
             ;
             eofile:  ; end of file, close destination
0151 11da01           lxi     d,dfcb   ; destination
0154 cd6e01           call    close    ; 255 if error
0157 21bb01           lxi     h,wrprot; ready message
015a 3c               inr     a        ; 255 becomes 00
015b cc6101           cz      finis    ; shouldn't happen
             ;
             ;        copy operation complete, end
```

```
015e 11cc01          lxi     d,normal; ready message
                ;
                finis:  ; write message given by de, reboot
0161 0e09            mvi     c,printf
0163 cd0500          call    bdos    ; write message
0166 c30000          jmp     boot    ; reboot system
                ;
                ;       system interface subroutines
                ;       (all return directly from bdos)
                ;
0169 0e0f       open:   mvi     c,openf
016b c30500          jmp     bdos
                ;
016e 0e10       close:  mvi     c,closef
0170 c30500          jmp     bdos
                ;
0173 0e13       delete: mvi     c,deletef
0175 c30500          jmp     bdos
                ;
0178 0e14       read:   mvi     c,readf
017a c30500          jmp     bdos
                ;
017d 0e15       write:  mvi     c,writef
017f c30500          jmp     bdos
                ;
0182 0e16       make:   mvi     c,makef
0184 c30500          jmp     bdos
                ;
                ;       console messages
0187 6e6f20fnofile: db      'no source file$'
0196 6e6f209nodir:  db      'no directory space$'
01a9 6f7574fspace:  db      'out of data space$'
01bb 7772695wrprot: db      'write protected?$'
01cc 636f700normal: db      'copy complete$'
                ;
                ;       data areas
01da            dfcb:   ds      33      ; destination fcb
01fa =          dfcbcr  equ     dfcb+32 ; current record
                ;
01fb                    ds      32      ; 16 level stack
                stack:
021b                    end
```

Note that there are several simplifications in this particular program. First, there are no checks for invalid file names which could, for example, contain ambiguous references. This situation could be detected by scanning the 32 byte default area starting at location 005CH for ASCII question marks. A check should also be made to ensure that the file names have, in fact, been included (check locations 005DH and 006DH for non-blank ASCII characters). Finally, a check should be made to ensure that the source and destination file names are different. A speed improvement could be made by buffering more data on each read operation. One could, for example, determine the size of memory by fetching FBASE from location 0006H and use the entire remaining portion of memory for a data buffer. In this case, the programmer simply resets the DMA address to the next successive 128 byte area before each read. Upon writing to the destination file, the DMA address is reset to the beginning of the buffer and incremented by 128 bytes to the end as each record is transferred to the destination file.

4. A SAMPLE FILE DUMP UTILITY.

The file dump program shown below is slightly more complex than the simple copy program given in the previous section. The dump program reads an input file, specified in the CCP command line, and displays the content of each record in hexadecimal format at the console. Note that the dump program saves the CCP's stack upon entry, resets the stack to a local area, and restores the CCP's stack before returning directly to the CCP. Thus, the dump program does not perform and warm start at the end of processing.

```
                ; DUMP program reads input file and displays hex data
                ;
0100                    org     100h
0005 =          bdos    equ     0005h   ;dos entry point
0001 =          cons    equ     1       ;read console
0002 =          typef   equ     2       ;type function
0009 =          printf  equ     9       ;buffer print entry
000b =          brkf    equ     11      ;break key function (true if char
000f =          openf   equ     15      ;file open
0014 =          readf   equ     20      ;read function
                ;
005c =          fcb     equ     5ch     ;file control block address
0080 =          buff    equ     80h     ;input disk buffer address
                ;
                ;       non graphic characters
000d =          cr      equ     0dh     ;carriage return
000a =          lf      equ     0ah     ;line feed
                ;
                ;       file control block definitions
005c =          fcbdn   equ     fcb+0   ;disk name
005d =          fcbfn   equ     fcb+1   ;file name
0065 =          fcbft   equ     fcb+9   ;disk file type (3 characters)
0068 =          fcbrl   equ     fcb+12  ;file's current reel number
006b =          fcbrc   equ     fcb+15  ;file's record count (0 to 128)
007c =          fcbcr   equ     fcb+32  ;current (next) record number (0
007d =          fcbln   equ     fcb+33  ;fcb length
                ;
                ;       set up stack
0100 210000             lxi     h,0
0103 39                 dad     sp
                ;       entry stack pointer in hl from the ccp
0104 221502             shld    oldsp
                ;       set sp to local stack area (restored at finis)
0107 315702             lxi     sp,stktop
                ;       read and print successive buffers
010a cdc101             call    setup   ;set up input file
010d feff               cpi     255     ;255 if file not present
010f c21b01             jnz     openok  ;skip if open is ok
                ;
                ;       file not there, give error message and return
0112 11f301             lxi     d,opnmsg
0115 cd9c01             call    err
0118 c35101             jmp     finis   ;to return
                ;
```

```
                openok: ;open operation ok, set buffer index to end
011b 3e80               mvi     a,80h
011d 321302             sta     ibp     ;set buffer pointer to 80h
                ;       hl contains next address to print
0120 210000             lxi     h,0     ;start with 0000
                ;
                gloop:
0123 e5                 push    h       ;save line position
0124 cda201             call    gnb
0127 e1                 pop     h       ;recall line position
0128 da5101             jc      finis   ;carry set by gnb if end file
012b 47                 mov     b,a
                ;       print hex values
                ;       check for line fold
012c 7d                 mov     a,l
012d e60f               ani     0fh     ;check low 4 bits
012f c24401             jnz     nonum
                ;       print line number
0132 cd7201             call    crlf
                ;
                ;       check for break key
0135 cd5901             call    break
                ;       accum lsb = 1 if character ready
0138 0f                 rrc             ;into carry
0139 da5101             jc      finis   ;don't print any more
                ;
013c 7c                 mov     a,h
013d cd8f01             call    phex
0140 7d                 mov     a,l
0141 cd8f01             call    phex
                nonum:
0144 23                 inx     h       ;to next line number
0145 3e20               mvi     a,' '
0147 cd6501             call    pchar
014a 78                 mov     a,b
014b cd8f01             call    phex
014e c32301             jmp     gloop
                ;
                finis:
                ;       end of dump, return to ccp
                ;       (note that a jmp to 0000h reboots)
0151 cd7201             call    crlf
0154 2a1502             lhld    oldsp
0157 f9                 sphl
                ;       stack pointer contains ccp's stack location
0158 c9                 ret             ;to the ccp
                ;
                ;
                ;       subroutines
                ;
                break:  ;check break key (actually any key will do)
0159 e5d5c5             push h! push d! push b; environment saved
015c 0e0b               mvi     c,brkf
015e cd0500             call    bdos
0161 c1d1e1             pop b! pop d! pop h; environment restored
```

```
0164 c9                 ret
              ;
              pchar:    ;print a character
0165 e5d5c5             push h! push d! push b; saved
0168 0e02               mvi     c,typef
016a 5f                 mov     e,a
016b cd0500             call    bdos
016e c1d1e1             pop b! pop d! pop h; restored
0171 c9                 ret
              ;
              crlf:
0172 3e0d               mvi     a,cr
0174 cd6501             call    pchar
0177 3e0a               mvi     a,lf
0179 cd6501             call    pchar
017c c9                 ret
              ;
              ;
              pnib:     ;print nibble in reg a
017d e60f               ani     0fh     ;low 4 bits
017f fe0a               cpi     10
0181 d28901             jnc     p10
              ;         less than or equal to 9
0184 c630               adi     '0'
0186 c38b01             jmp     prn
              ;
              ;         greater or equal to 10
0189 c637     p10:      adi     'a' - 10
018b cd6501   prn:      call    pchar
018e c9                 ret
              ;
              phex:     ;print hex char in reg a
018f f5                 push    psw
0190 0f                 rrc
0191 0f                 rrc
0192 0f                 rrc
0193 0f                 rrc
0194 cd7d01             call    pnib    ;print nibble
0197 f1                 pop     psw
0198 cd7d01             call    pnib
019b c9                 ret
              ;
              err:      ;print error message
              ;         d,e addresses message ending with "$"
019c 0e09               mvi     c,printf        ;print buffer function
019e cd0500             call    bdos
01a1 c9                 ret
              ;
              ;
              gnb:      ;get next byte
01a2 3a1302             lda     ibp
01a5 fe80               cpi     80h
01a7 c2b301             jnz     g0
              ;         read another buffer
              ;
```

```
              ;
0laa cdce01             call    diskr
0lad b7                 ora     a       ;zero value if read ok
0lae cab301             jz      g0      ;for another byte
              ;         end of data, return with carry set for eof
0lbl 37                 stc
0lb2 c9                 ret
              ;
              g0:       ;read the byte at buff+reg a
0lb3 5f                 mov     e,a     ;ls byte of buffer index
0lb4 1600               mvi     d,0     ;double precision index to de
0lb6 3c                 inr     a       ;index=index+1
0lb7 321302             sta     ibp     ;back to memory
              ;         pointer is incremented
              ;         save the current file address
0lba 218000             lxi     h,buff
0lbd 19                 dad     d
              ;         absolute character address is in hl
0lbe 7e                 mov     a,m
              ;         byte is in the accumulator
0lbf b7                 ora     a       ;reset carry bit
0lc0 c9                 ret
              ;
              setup:    ;set up file
              ;         open the file for input
0lcl af                 xra     a       ;zero to accum
0lc2 327c00             sta     fcbcr   ;clear current record
              ;
0lc5 115c00             lxi     d,fcb
0lc8 0e0f               mvi     c,openf
0lca cd0500             call    bdos
              ;         255 in accum if open error
0lcd c9                 ret
              ;
              diskr:    ;read disk file record
0lce e5d5c5             push h! push d! push b
0ldl 115c00             lxi     d,fcb
0ld4 0el4               mvi     c,readf
0ld6 cd0500             call    bdos
0ld9 cldlel             pop b! pop d! pop h
0ldc c9                 ret
              ;
              ;         fixed message area
0ldd 46494c0signon: db          'file dump version 2.0$'
0lf3 0d0a4e0opnmsg: db          cr,lf,'no input file present on disk$'

              ;         variable area
0213          ibp:      ds      2       ;input buffer pointer
0215          oldsp:    ds      2       ;entry sp value from ccp
              ;
              ;         stack area
0217                    ds      64      ;reserve 32 level stack
              stktop:
              ;
0257                    end
```

5. A SAMPLE RANDOM ACCESS PROGRAM.

This manual is concluded with a rather extensive, but complete example of random access operation. The program listed below performs the simple function of reading or writing random records upon command from the terminal. Given that the program has been created, assembled, and placed into a file labelled RANDOM.COM, the CCP level command:

RANDOM X.DAT

starts the test program. The program looks for a file by the name X.DAT (in this particular case) and, if found, proceeds to prompt the console for input. If not found, the file is created before the prompt is given. Each prompt takes the form

next command?

and is followed by operator input, terminated by a carriage return. The input commands take the form

nW nR Q

where n is an integer value in the range Ø to 65535, and W, R, and Q are simple command characters corresponding to random write, random read, and quit processing, respectively. If the W command is issued, the RANDOM program issues the prompt

type data:

The operator then responds by typing up to 127 characters, followed by a carriage return. RANDOM then writes the character string into the X.DAT file at record n. If the R command is issued, RANDOM reads record number n and displays the string value at the console. If the Q command is issued, the X.DAT file is closed, and the program returns to the console command processor. In the interest of brevity, the only error message is

error, try again

The program begins with an initialization section where the input file is opened or created, followed by a continuous loop at the label "ready" where the individual commands are interpreted. The default file control block at ØØ5CH and the default buffer at ØØ8ØH are used in all disk operations. The utility subroutines then follow, which contain the principal input line processor, called "readc." This particular program shows the elements of random access processing, and can be used as the basis for further program development.

```
                ;***************************************************
                ;*                                                 *
                ;* sample random access program for cp/m 2.0       *
                ;*                                                 *
                ;***************************************************
0100                    org     100h    ;base of tpa
                ;
0000 =          reboot  equ     0000h   ;system reboot
0005 =          bdos    equ     0005h   ;bdos entry point
                ;
0001 =          coninp  equ     1       ;console input function
0002 =          conout  equ     2       ;console output function
0009 =          pstring equ     9       ;print string until '$'
000a =          rstring equ     10      ;read console buffer
000c =          version equ     12      ;return version number
000f =          openf   equ     15      ;file open function
0010 =          closef  equ     16      ;close function
0016 =          makef   equ     22      ;make file function
0021 =          readr   equ     33      ;read random
0022 =          writer  equ     34      ;write random
                ;
005c =          fcb     equ     005ch   ;default file control block
007d =          ranrec  equ     fcb+33  ;random record position
007f =          ranovf  equ     fcb+35  ;high order (overflow) byte
0080 =          buff    equ     0080h   ;buffer address
                ;
000d =          cr      equ     0dh     ;carriage return
000a =          lf      equ     0ah     ;line feed
                ;
                ;***************************************************
                ;*                                                 *
                ;* load SP, set-up file for random access          *
                ;*                                                 *
                ;***************************************************
0100 31bc0              lxi     sp,stack
                ;
                ;       version 2.0?
0103 0e0c               mvi     c,version
0105 cd050              call    bdos
0108 fe20               cpi     20h     ;version 2.0 or better?
010a d2160              jnc     versok
                ;       bad version, message and go back
010d 111b0              lxi     d,badver
0110 cdda0              call    print
0113 c3000              jmp     reboot
                ;
                versok:
                ;       correct version for random access
0116 0e0f               mvi     c,openf ;open default fcb
0118 115c0              lxi     d,fcb
011b cd050              call    bdos
011e 3c                 inr     a       ;err 255 becomes zero
011f c2370              jnz     ready
                ;
                ;       cannot open file, so create it
```

```
0122 0e16          mvi     c,makef
0124 115c0         lxi     d,fcb
0127 cd050         call    bdos
012a 3c            inr     a       ;err 255 becomes zero
012b c2370         jnz     ready
               ;
               ;       cannot create file, directory full
012e 113a0         lxi     d,nospace
0131 cdda0         call    print
0134 c3000         jmp     reboot  ;back to ccp
               ;
               ;***********************************************
               ;*                                             *
               ;*  loop back to "ready" after each command    *
               ;*                                             *
               ;***********************************************
               ;
               ready:
               ;       file is ready for processing
               ;
0137 cde50         call    readcom ;read next command
013a 227d0         shld    ranrec  ;store input record#
013d 217f0         lxi     h,ranovf
0140 3600          mvi     m,0     ;clear high byte if set
0142 fe51          cpi     'Q'     ;quit?
0144 c2560         jnz     notq
               ;
               ;       quit processing, close file
0147 0e10          mvi     c,closef
0149 115c0         lxi     d,fcb
014c cd050         call    bdos
014f 3c            inr     a       ;err 255 becomes 0
0150 cab90         jz      error   ;error message, retry
0153 c3000         jmp     reboot  ;back to ccp
               ;
               ;***********************************************
               ;*                                             *
               ;* end of quit command, process write          *
               ;*                                             *
               ;***********************************************
               notq:
               ;       not the quit command, random write?
0156 fe57          cpi     'W'
0158 c2890         jnz     notw
               ;
               ;       this is a random write, fill buffer until cr
015b 114d0         lxi     d,datmsg
015e cdda0         call    print   ;data prompt
0161 0e7f          mvi     c,127   ;up to 127 characters
0163 21800         lxi     h,buff  ;destination
               rloop:  ;read next character to buff
0166 c5            push    b       ;save counter
0167 e5            push    h       ;next destination
0168 cdc20         call    getchr  ;character to a
016b e1            pop     h       ;restore counter
```

```
016c c1             pop     b       ;restore next to fill
016d fe0d           cpi     cr      ;end of line?
016f ca780          jz      erloop
              ;     not end, store character
0172 77             mov     m,a
0173 23             inx     h       ;next to fill
0174 0d             dcr     c       ;counter goes down
0175 c2660          jnz     rloop   ;end of buffer?
              erloop:
              ;     end of read loop, store 00
0178 3600           mvi     m,0
              ;
              ;     write the record to selected record number
017a 0e22           mvi     c,writer
017c 115c0          lxi     d,fcb
017f cd050          call    bdos
0182 b7             ora     a       ;error code zero?
0183 c2b90          jnz     error   ;message if not
0186 c3370          jmp     ready   ;for another record
              ;
              ;*****************************************************
              ;*                                                   *
              ;* end of write command, process read                *
              ;*                                                   *
              ;*****************************************************
              notw:
              ;     not a write command, read record?
0189 fe52           cpi     'R'
018b c2b90          jnz     error   ;skip if not
              ;
              ;     read random record
018e 0e21           mvi     c,readr
0190 115c0          lxi     d,fcb
0193 cd050          call    bdos
0196 b7             ora     a       ;return code 00?
0197 c2b90          jnz     error
              ;
              ;     read was successful, write to console
019a cdcf0          call    crlf    ;new line
019d 0e80           mvi     c,128   ;max 128 characters
019f 21800          lxi     h,buff  ;next to get
              wloop:
01a2 7e             mov     a,m     ;next character
01a3 23             inx     h       ;next to get
01a4 e67f           ani     7fh     ;mask parity
01a6 ca370          jz      ready   ;for another command if 00
01a9 c5             push    b       ;save counter
01aa e5             push    h       ;save next to get
01ab fe20           cpi     ' '     ;graphic?
01ad d4c80          cnc     putchr  ;skip output if not
01b0 e1             pop     h
01b1 c1             pop     b
01b2 0d             dcr     c       ;count=count-1
01b3 c2a20          jnz     wloop
01b6 c3370          jmp     ready
```

```
          ;
          ;***********************************************
          ;*                                             *
          ;* end of read command, all errors end-up here *
          ;*                                             *
          ;***********************************************
          ;
          error:
01b9 11590          lxi     d,errmsg
01bc cdda0          call    print
01bf c3370          jmp     ready
          ;
          ;***********************************************
          ;*                                             *
          ;* utility subroutines for console i/o         *
          ;*                                             *
          ;***********************************************
          getchr:
                    ;read next console character to a
01c2 0e01           mvi     c,coninp
01c4 cd050          call    bdos
01c7 c9             ret
          ;
          putchr:
                    ;write character from a to console
01c8 0e02           mvi     c,conout
01ca 5f             mov     e,a     ;character to send
01cb cd050          call    bdos    ;send character
01ce c9             ret
          ;
          crlf:
                    ;send carriage return line feed
01cf 3e0d           mvi     a,cr    ;carriage return
01d1 cdc80          call    putchr
01d4 3e0a           mvi     a,lf    ;line feed
01d6 cdc80          call    putchr
01d9 c9             ret
          ;
          print:
                    ;print the buffer addressed by de until $
01da d5             push    d
01db cdcf0          call    crlf
01de d1             pop     d       ;new line
01df 0e09           mvi     c,pstring
01e1 cd050          call    bdos    ;print the string
01e4 c9             ret
          ;
          readcom:
                    ;read the next command line to the conbuf
01e5 116b0          lxi     d,prompt
01e8 cdda0          call    print   ;command?
01eb 0e0a           mvi     c,rstring
01ed 117a0          lxi     d,conbuf
01f0 cd050          call    bdos    ;read command line
          ;         command line is present, scan it
```

```
Ø1f3 21ØØØ          lxi     h,Ø       ;start with ØØØØ
Ø1f6 117cØ          lxi     d,conlin;command line
Ø1f9 1a     readc:  ldax    d         ;next command character
Ø1fa 13             inx     d         ;to next command position
Ø1fb b7             ora     a         ;cannot be end of command
Ø1fc c8             rz
            ;       not zero, numeric?
Ø1fd d63Ø           sui     'Ø'
Ø1ff feØa           cpi     1Ø        ;carry if numeric
Ø2Ø1 d213Ø          jnc     endrd
            ;       add-in next digit
Ø2Ø4 29             dad     h         ;*2
Ø2Ø5 4d             mov     c,l
Ø2Ø6 44             mov     b,h       ;bc = value * 2
Ø2Ø7 29             dad     h         ;*4
Ø2Ø8 29             dad     h         ;*8
Ø2Ø9 Ø9             dad     b         ;*2 + *8 = *1Ø
Ø2Øa 85             add     l         ;+digit
Ø2Øb 6f             mov     l,a
Ø2Øc d2f9Ø          jnc     readc     ;for another char
Ø2Øf 24             inr     h         ;overflow
Ø21Ø c3f9Ø          jmp     readc     ;for another char
            endrd:
            ;       end of read, restore value in a
Ø213 c63Ø           adi     'Ø'       ;command
Ø215 fe61           cpi     'a'       ;translate case?
Ø217 d8             rc
            ;       lower case, mask lower case bits
Ø218 e65f           ani     1Ø1$1111b
Ø21a c9             ret
            ;
            ;*****************************************************
            ;*                                                   *
            ;* string data area for console messages             *
            ;*                                                   *
            ;*****************************************************
            badver:
Ø21b 536f79         db      'sorry, you need cp/m version 2$'
            nospace:
Ø23a 4e6f29         db      'no directory space$'
            datmsg:
Ø24d 54797Ø         db      'type data: $'
            errmsg:
Ø259 457272         db      'error, try again.$'
            prompt:
Ø26b 4e657Ø         db      'next command? $'
            ;
```

```
                ;*************************************************
                ;*                                               *
                ;* fixed and variable data area                  *
                ;*                                               *
                ;*************************************************
027a 21         conbuf: db      conlen  ;length of console buffer
027b            consiz: ds      1       ;resulting size after read
027c            conlin: ds      32      ;length 32 buffer
0021 =          conlen  equ     $-consiz
                ;
029c                    ds      32      ;16 level stack
                stack:
02bc                    end
```

Again, major improvements could be made to this particular program to enhance its operation. In fact, with some work, this program could evolve into a simple data base management system. One could, for example, assume a standard record size of 128 bytes, consisting of arbitrary fields within the record. A program, called GETKEY, could be developed which first reads a sequential file and extracts a specific field defined by the operator. For example, the command

GETKEY NAMES.DAT LASTNAME 10 20

would cause GETKEY to read the data base file NAMES.DAT and extract the "LASTNAME" field from each record, starting at position 10 and ending at character 20. GETKEY builds a table in memory consisting of each particular LASTNAME field, along with its 16-bit record number location within the file. The GETKEY program then sorts this list, and writes a new file, called LASTNAME.KEY, which is an alphabetical list of LASTNAME fields with their corresponding record numbers. (This list is called an "inverted index" in information retrieval parlance.)

Rename the program shown above as QUERY, and massage it a bit so that it reads a sorted key file into memory. The command line might appear as:

QUERY NAMES.DAT LASTNAME.KEY

Instead of reading a number, the QUERY program reads an alphanumeric string which is a particular key to find in the NAMES.DAT data base. Since the LASTNAME.KEY list is sorted, you can find a particular entry quite rapidly by performing a "binary search," similar to looking up a name in the telephone book. That is, starting at both ends of the list, you examine the entry halfway in between and, if not matched, split either the upper half or the lower half for the next search. You'll quickly reach the item you're looking for (in log2(n) steps) where you'll find the corresponding record number. Fetch and display this record at the console, just as we have done in the program shown above.

At this point you're just getting started. With a little more work, you can allow a fixed grouping size which differs from the 128 byte record shown above. This is accomplished by keeping track of the record number as well as the byte offset within the record. Knowing the group size, you randomly access the record containing the proper group, offset to the beginning of the group within the record read sequentially until the group size has been exhausted.

Finally, you can improve QUERY considerably by allowing boolean expressions which compute the set of records which satisfy several relationships, such as a LASTNAME between HARDY and LAUREL, and an AGE less than 45. Display all the records which fit this description. Finally, if your lists are getting too big to fit into memory, randomly access your key files from the disk as well. One note of consolation after all this work: if you make it through the project, you'll have no more need for this manual!

6. SYSTEM FUNCTION SUMMARY.

FUNC	FUNCTION NAME	INPUT PARAMETERS	OUTPUT RESULTS
Ø	System Reset	none	none
1	Console Input	none	A = char
2	Console Output	E = char	none
3	Reader Input	none	A = char
4	Punch Output	E = char	none
5	List Output	E = char	none
6	Direct Console I/O	see def	see def
7	Get I/O Byte	none	A = IOBYTE
8	Set I/O Byte	E = IOBYTE	none
9	Print String	DE = .Buffer	none
1Ø	Read Console Buffer	DE = .Buffer	see def
11	Get Console Status	none	A = ØØ/FF
12	Return Version Number	none	HL= Version*
13	Reset Disk System	none	see def
14	Select Disk	E = Disk Number	see def
15	Open File	DE = .FCB	A = Dir Code
16	Close File	DE = .FCB	A = Dir Code
17	Search for First	DE = .FCB	A = Dir Code
18	Search for Next	none	A = Dir Code
19	Delete File	DE = .FCB	A = Dir Code
2Ø	Read Sequential	DE = .FCB	A = Err Code
21	Write Sequential	DE = .FCB	A = Err Code
22	Make File	DE = .FCB	A = Dir Code
23	Rename File	DE = .FCB	A = Dir Code
24	Return Login Vector	none	HL= Login Vect*
25	Return Current Disk	none	A = Cur Disk#
26	Set DMA Address	DE = .DMA	none
27	Get Addr(Alloc)	none	HL= .Alloc
28	Write Protect Disk	none	see def
29	Get R/O Vector	none	HL= R/O Vect*
3Ø	Set File Attributes	DE = .FCB	see def
31	Get Addr(disk parms)	none	HL= .DPB
32	Set/Get User Code	see def	see def
33	Read Random	DE = .FCB	A = Err Code
34	Write Random	DE = .FCB	A = Err Code
35	Compute File Size	DE = .FCB	rØ, rl, r2
36	Set Random Record	DE = .FCB	rØ, rl, r2

* Note that A = L, and B = H upon return

DIGITAL RESEARCH®

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

CP/M ASSEMBLER (ASM)

USER'S GUIDE

Table of Contents

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.p1p2p3
```

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 re-
                      ceive the hex file
     Z                skips the generation of the hex file
p3:  A,B, ..., Y      designates the disk name which will re-
                      ceive 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

-embly 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 a 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 (alphabetics 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 alphabetics become 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$name
x:	yxl:	longer$named$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 compatability 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 withing 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 logicl 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 unparenthesize 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 parenthesize 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 assmbly, 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
CRTBASE   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 CRTBASE
CONIN     EQU   CRTBASE      ;CONSOLE INPUT
CONOUT    EQU   CRTBASE+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,5,377Q,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 forst 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 breifly 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  al      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 repetoire 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 Ø	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 ;      SORT PROGRAM IN CP/M ASSEMBLY LANGUAGE
                    ;        START AT THE BEGINNING OF THE TRANSIENT PROGRAM AR
 0100                        ORG     100H
      generated machine code
 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
                  ;
                  ;          COMPARE I WITH ARRAY SIZE
 010A 7E          COMP:      MOV     A,M       ;A REGISTER = I
 010B FE09                   CPI     N-1       ;CY SET IF I < (N-1)
 010D D21901                 JNC     CONT      ;CONTINUE IF I <= (N-2)
                  ;
                  ;          END OF ONE PASS THROUGH DATA
 0110 214601                 LXI     H,SW      ;CHECK FOR ZERO SWITCHES
 0113 7EB7C20001             MOV A,M! ORA A! JNZ SORT ;END OF SORT IF SW=0
                  ;
 0118 FF                     RST 7             ;GO TO THE DEBUGGER INSTEAD OF RE
                  ; truncated
                  ;          CONTINUE THIS PASS
                  ;          ADDRESSING I, SO LOAD AV(I) INTO REGISTERS
 0119 5F16002148CONT:        MOV E,A! MVI D,0! LXI H,AV! DAD D! DAD D
 0121 4E792346               MOV C,M! MOV A,C! INX H! MOV B,M
                  ;          LOW ORDER BYTE IN A AND C, HIGH ORDER BYTE IN B
                  ;
                  ;          MOV H AND L TO ADDRESS AV(I+1)
 0125 23                     INX     H
                  ;
                  ;          COMPARE VALUE WITH REGS CONTAINING AV(I)
 0126 965778239E             SUB M! MOV D,A! MOV A,B! INX H! SBB M    ;SUBTRACT
                  ;
                  ;          BORROW SET IF AV(I+1) > AV(I)
 012B DA3F01                 JC      INCI      ;SKIP IF IN PROPER ORDER
                  ;
                  ;          CHECK FOR EQUAL VALUES
 012E B2CA3F01               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
                  ;
                  ;     INCREMENT SWITCH COUNT
 013B 21460134          LXI H,SW! INR M
                  ;
                  ;     INCREMENT I
 013F 21470134C3INCI:   LXI H,I! INR M! JMP COMP
                  ;
                  ;     DATA DEFINITION SECTION
 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
:100110002146017EB7C20001FF5F16002148011983
:10012000194E79234623965778239EDA3F01B2CAA7
:100130003F0156702B5E712B722B732146013421C7
:07014000470134C30A01006E
:10014800050064001E00320014000700E8032C01BB
:0401580064000180BE
:0000000000
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
                                                                   abort with
                                                                   rubout
C0Z0M0E0I0 A=00 B=0000 D=0000 H=0000 S=0100 P=0100 LXI  H,0146*0100
-T10  trace 10_16 steps

C0Z0M0E0I0 A=01 B=0000 D=0000 H=0146 S=0100 P=0100 LXI  H,0146
C0Z0M0E0I0 A=01 B=0000 D=0000 H=0146 S=0100 P=0103 MVI  M,01
C0Z0M0E0I0 A=01 B=0000 D=0000 H=0146 S=0100 P=0105 LXI  H,0147
C0Z0M0E0I0 A=01 B=0000 D=0000 H=0147 S=0100 P=0108 MVI  M,00
C0Z0M0E0I0 A=01 B=0000 D=0000 H=0147 S=0100 P=010A MOV  A,M
C0Z0M0E0I0 A=00 B=0000 D=0000 H=0147 S=0100 P=010B CPI  09
C1Z0M1E0I0 A=00 B=0000 D=0000 H=0147 S=0100 P=010D JNC  0119
C1Z0M1E0I0 A=00 B=0000 D=0000 H=0147 S=0100 P=0110 LXI  H,0146
C1Z0M1E0I0 A=00 B=0000 D=0000 H=0146 S=0100 P=0113 MOV  A,M
C1Z0M1E0I0 A=01 B=0000 D=0000 H=0146 S=0100 P=0114 ORA  A
C0Z0M0E0I0 A=01 B=0000 D=0000 H=0146 S=0100 P=0115 JNZ  0100
C0Z0M0E0I0 A=01 B=0000 D=0000 H=0146 S=0100 P=0100 LXI  H,0146
C0Z0M0E0I0 A=01 B=0000 D=0000 H=0146 S=0100 P=0103 MVI  M,01
C0Z0M0E0I0 A=01 B=0000 D=0000 H=0146 S=0100 P=0105 LXI  H,0147
C0Z0M0E0I0 A=01 B=0000 D=0000 H=0147 S=0100 P=0108 MVI  M,00
C0Z0M0E0I0 A=01 B=0000 D=0000 H=0147 S=0100 P=010A MOV  A,M*010B
-A10D

010D  JC 119  change to a jump on carry                  stopped at
0110                                                       10BH
```

machine code in HEX format

```
-XP
P=010B 100   reset program counter back to beginning of program
-T10   trace execution for 10H steps

C0Z0M0E0I0 A=00 B=0000 D=0000 H=0147 S=0100 P=0100 LXI  H,0146
C0Z0M0E0I0 A=00 B=0000 D=0000 H=0146 S=0100 P=0103 MVI  M,01
C0Z0M0E0I0 A=00 B=0000 D=0000 H=0146 S=0100 P=0105 LXI  H,0147
C0Z0M0E0I0 A=00 B=0000 D=0000 H=0147 S=0100 P=0108 MVI  M,00
C0Z0M0E0I0 A=00 B=0000 D=0000 H=0147 S=0100 P=010A MOV  A,M
C0Z0M0E0I0 A=00 B=0000 D=0000 H=0147 S=0100 P=010B CPI  09
C1Z0M1E0I0 A=00 B=0000 D=0000 H=0147 S=0100 P=010D JC   0119   Altered instruction
C1Z0M1E0I0 A=00 B=0000 D=0000 H=0147 S=0100 P=0119 MOV  E,A
C1Z0M1E0I0 A=00 B=0000 D=0000 H=0147 S=0100 P=011A MVI  D,00
C1Z0M1E0I0 A=00 B=0000 D=0000 H=0147 S=0100 P=011C LXI  H,0148
C1Z0M1E0I0 A=00 B=0000 D=0000 H=0148 S=0100 P=011F DAD  D
C0Z0M1E0I0 A=00 B=0000 D=0000 H=0148 S=0100 P=0120 DAD  D
C0Z0M1E0I0 A=00 B=0000 D=0000 H=0148 S=0100 P=0121 MOV  C,M
C0Z0M1E0I0 A=00 B=0005 D=0000 H=0148 S=0100 P=0122 MOV  A,C
C0Z0M1E0I0 A=05 B=0005 D=0000 H=0148 S=0100 P=0123 INX  H
C0Z0M1E0I0 A=05 B=0005 D=0000 H=0149 S=0100 P=0124 MOV  B,M*0125
-L100                                                    Automatic
                                                         breakpoint
0100  LXI  H,0146
0103  MVI  M,01
0105  LXI  H,0147
0108  MVI  M,00
010A  MOV  A,M      list some code
010B  CPI  09       from 100H
010D  JC   0119
0110  LXI  H,0146
0113  MOV  A,M
0114  ORA  A
0115  JNZ  0100
-L

0118  RST  07
0119  MOV  E,A      list more
011A  MVI  D,00
011C  LXI  H,0148
- abort list with rubout
-G,11B   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
-T4   look at looping program in trace mode          looping indefinitely)

C0Z0M0E0I0 A=38 B=0064 D=0006 H=0156 S=0100 P=0127 MOV  D,A
C0Z0M0E0I0 A=38 B=0064 D=3806 H=0156 S=0100 P=0128 MOV  A,B
C0Z0M0E0I0 A=00 B=0064 D=3806 H=0156 S=0100 P=0129 INX  H
C0Z0M0E0I0 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 ................
```

```
-G0  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 ctl-C (G0 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 P.C=100H

*0118  programmed stop (RST 7) encountered
-D148
                                   data properly sorted
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 ................
0170 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................

-G0  return to CP/M
```

```
ED SORT.ASM    make changes to original program
 ctl-Z
*N,0^Z0TT    find next ",0"
        MVI       M,0       ;I = 0
*-    up one line in text
        LXI       H,I       ;ADDRESS INDEX
*-    up another line
        MVI       M,1       ;SET TO 1 FOR FIRST ITERATION
*KT    kill line and type next line
        LXI       H,I       ;ADDRESS INDEX
*I    insert new line
        MVI       M,0       ;ZERO SW
*T
        LXI       H,I       ;ADDRESS INDEX
*NJNC^Z0T
        JNC*T
        CONT    ;CONTINUE IF I <= (N-2)
*-2DIC^Z0LT
        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

015C  next address to assemble
003H USE FACTOR
END OF ASSEMBLY

DDT SORT.HEX    test program changes

16K DDT VER 1.0
NEXT  PC
015C 0000
-G100

*0118
-D148
                                 data sorted
0148 05 00 07 00 14 00 1E 00 ........
0150 32 00 64 00 64 00 2C 01 E8 03 01 80 00 08 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

-G0    return to CP/M - program checks OK.
```

MACRO - 80
Assembler
Reference Manual

NOTE: Chapters 2 and 3 of this manual have been deleted as the associated software is not provided as part of the Xerox software.

8401-340-03

Microsoft MACRO-80 Assembler, Release 3.4 November 1980

ADDENDA TO: Utility Software Manual
MACRO-80 Assembler Reference Manual
XMACRO-86 Assembler Reference Manual

The following features were added or modified in release 3.4.

Add to Section 2.2.2 Switches

Switch	Action
/M	Initialize Block Data Areas. If the programmer wants the area that is defined by the DS (Define Space) pseudo-op initialized to zeros, then the programmer should use the /M switch in the command line. Otherwise, the space is not guaranteed to contain zeros. That is, DS does not automatically initialize the space to zeros.
/X	The presence or absence of /X in the command line sets the initial current mode and the initial value of the default for listing or suppressing lines in false conditional blocks. /X sets the current mode and initial value of default to not-to-list. No /X sets current mode and initial value of default to list. Current mode determines whether false conditionals will be listed or suppressed. The initial value of the default is used with the .TFCOND pseudo-op so that .TFCOND is independent of .SFCOND and .LFCOND. If the program contains .SFCOND or .LFCOND, /X has no effect after .SFCOND or .LFCOND is encountered until a .TFCOND is encountered in the file. So /X has an effect only when used with a file that contains no conditional listing pseudo-ops or when used with .TFCOND.

The following chart illustrates the effects of the three pseudo-ops when encountered under /X and under no /X. See the addition to Section 2.6.27 below for a full description of the three conditional listing pseudo-ops.

PSEUDO-OP	NO /X	/X
(none)	ON	OFF
.	.	.
.	.	.
.	.	.
.SFCOND	OFF	OFF
.	.	.
.	.	.
.	.	.
.LFCOND	ON	ON
.	.	.
.	.	.
.	.	.
.TFCOND	OFF	ON
.	.	.
.	.	.
.	.	.
.TFCOND	ON	OFF
.	.	.
.	.	.
.	.	.
.SFCOND	OFF	OFF
.	.	.
.	.	.
.	.	.
.TFCOND	OFF	ON
.TFCOND	ON	OFF
.	.	.
.	.	.
.	.	.
.TFCOND	OFF	ON

Add to Section 2.6.26 Conditional Pseudo Operations

IFIDN <arg1>,<arg2> — True if the string <arg1> is IDeNtical to the string <arg2>. The angle brackets around <arg1> and <arg2> are required.

IFDIF <arg1>,<arg2> — True if the string <arg1> is DIFferent from the string <arg2>. The angle brackets around <arg1> and <arg2> are required.

Add to Section 2.6.27 Listing Control Pseudo Operations

There are now five listing control pseudo-ops. Output to the listing file can be controlled by the following pseudo-ops:

.LIST, .XLIST, .SFCOND, .LFCOND, .TFCOND

The three new pseudo-ops control the listing of conditional pseudo-op blocks which evaluate as false. These pseudo-ops give the programmer control over four cases.

1. Normally list false conditionals
   For this case, the programmer simply allows the default mode to control the listing. The default mode is list false conditionals. If the programmer decides to suppress false conditionals, the /X switch can be issued in the command line instead of editing the source file.

2. Normally suppress false conditionals
   For this case, the programmer issues the .TFCOND pseudo-op in the program file. .TFCOND reverses (toggles) the default, causing false conditionals to be suppressed. If the programmer decides to list false conditionals, the /X switch can be issued in the command line instead of editing the source file.

3. Always suppress/list false conditionals
   For these cases, the programmer issues either the .SFCOND pseudo-op to suppress false conditionals, or the .LFCOND pseudo-op to list all false conditionals.

4. Suppress/list some false conditionals
   For this case, the programmer has decided for most false conditionals whether to list or suppress, but for some false conditionals the programmer has not yet decided. For the false conditionals decided about, use .SFCOND or .LFCOND. For those not yet decided, use .TFCOND. .TFCOND sets the current and default settings to the opposite of the default. Initially, the default is set by giving /X or no /X in the command line. Two subcases exist:

   1. The programmer wants some false conditionals not to list unless /X is given. The programmer uses the .SFCOND and .LFCOND pseudo-ops to control which areas always suppress or list false conditionals. To selectively suppress some false conditionals, the programmer issues .TFCOND at the beginning of the conditional block and again at the end of the conditional block. (NOTE: The second .TFCOND is should be so that the default setting will be the same as the initial setting. Leaving the default equal

to the initial setting makes it easier to keep track of the default mode if there are many such areas.) If the conditional block evaluates as false, the lines will be suppressed. In this subcase, issuing the /X switch in the command line causes the conditional block affected by .TFCOND to list even if it evaluates as false.

2. The programmer wants some false conditionals to list unless /X is given. of the file. Two consecutive .TFCONDs places the conditional listing setting in initial state which is determined by the presence or absence of the /X switch (the first .TFCOND sets the default to not initial; the second to initial). The selected conditional block then responds to the /X switch: if a /X switch is issued in the command line, the conditional block is suppressed if false; if no /X switch is issued in the command line, the conditional block is listed even if false.

   The programmer then must reissue the .SFCOND or .LFCOND conditional listing pseudo-op to restore the suppress or list mode. Simply issuing another .TFCOND will not restore the prior mode, but will toggle the default setting. Since in this subcase, the next area of code is supposed to list or suppress false conditionals always, the programmer must issue .SFCOND or .LFCOND.

The three conditional listing pseudo-ops are summarized below.

PSEUDO-OP	DEFINITION
.SFCOND	Suppresses the listing of conditional blocks that evaluate as false.
.LFCOND	Restores the listing of conditional blocks that evaluate as false.
.TFCOND	Toggles the current setting which controls the listing false conditionals. .TFCOND sets the current and default setting to not default. If a /X switch is given in the MACRO-80 run command line for a file which contains .TFCOND, /X reverses the effect of .TFCOND.

Add to Section 2.7.9 Special Macro Operators and Forms

% The percent sign is used only in a macro argument. % converts the expression that follows it (usually a symbol) to a number in the current radix. During macro expansion, the number derived from converting the expression is substituted for the dummy. Using the % special operator allows a macro call by value. (Usually, a macro call is a call by reference with the text of the macro argument substituting exactly for the dummy.)

The expression following the % must conform to the same rules as the DS (Define Space) pseudo-op. A valid expression returning a non-relocatable constant is required.

EXAMPLE:

Normally, LB, the argument to MAKLAB, would be substituted for Y, the argument to MACRO, as a string. The % causes LB to be converted to a non-relocatable constant which is then substituted for Y. Without the % special operator, the result of assembly would be 'Error LB' rather than 'Error 1', etc.

```
MAKLAB   MACRO    Y
ERR&Y:   DB       'Error &Y',0
         ENDM
MAKERR   MACRO    X
LB       SET      0
         REPT     X
LB       SET      LB+1
         MAKLAB   %LB
         ENDM
         ENDM
```

When called by MAKERR 3, the assembler will generate:

```
ERR1:  DB    'Error 1',0
ERR2:  DB    'Error 2',0
ERR3:  DB    'Error 3',0
```

CONTENTS

This manual is designed to serve as a reference guide to the MACRO-80 package. It defines, explains and gives examples of all the features in MACRO-80 in terms that should be understandable to anyone familiar with assembly language programming. It is not intended, however, to serve as instructional material and presumes the user has substantial knowledge of assembly language programming. The user should refer to instructional material available from a variety of sources for additional tutorial information.

CHAPTER 2

MACRO-80 ASSEMBLER

2.1 RUNNING MACRO-80

The command to run MACRO-80 is

```
M80
```

MACRO-80 returns the prompt "*", indicating it is ready to accept commands.

NOTE

> If you are using the TEKDOS operating system, see Appendix A for proper command formats.

2.2 COMMAND FORMAT

A command to MACRO-80 consists of a string of filenames with optional switches. All filenames should follow the operating system's conventions for filenames and extensions. The default extensions supplied by Microsoft software are as follows:

File	CP/M	ISIS-II
Relocatable object file	REL	REL
Listing file	PRN	LST
MACRO-80 source file	MAC	MAC
FORTRAN source file	FOR	FOR
COBOL source	COB	COB
Absolute file	COM	

A command to MACRO-80 conveys the name of the source file to be assembled, the names of the file(s) to be created, and which assembly options are desired. The format of a MACRO-80 command is:

```
objfile,lstfile=source file
```

Only the equal sign and the source file field are required to create a relocatable object file with the default (source) filename and the default extension REL.

Otherwise, an object file is created only if the objfile field is filled, and a listing file is created only if the lstfile field is filled.

To assemble the source file without producing an object file or listing file, place only a comma to the left of the equal sign. This is a handy procedure that lets you check for syntax errors before assembling to an object file.

Examples:

*=TEST	Assemble the source file TEST.MAC and place the object file in TEST.REL.
*,=TEST	Assemble the source file TEST.MAC without creating an object or listing file. Useful for error checking.
TEST,TEST=TEST	Assemble the source file TEST.MAC, placing the object file in TEST.REL and the listing file in TEST.PRN. (With ISIS-II, the listing file is TEST.LST.)
*OBJECT=TEST	Assemble the source file TEST.MAC and place the object file in OBJECT.REL.
OBJECT,LIST=TEST	Assemble the source file TEST.MAC, placing the object file in OBJECT.REL and the listing file in LIST.PRN. (With ISIS-II, the listing file is LIST.LST.)

MACRO-80 also supports command lines; that is, the invocation and command may be typed on the same line. For example:

```
M80 ,=TEST
```

2.2.1 Devices

Any field in the MACRO-80 command string can also specify a device name. The default device name with the CP/M operating system is the currently logged disk. The default device name with the ISIS-II operating system is disk drive 0. The command format is:

```
dev:objfile,dev:lstfile=dev:source file
```

The device names are as follows:

Device	CP/M	ISIS-II
Disk drives	A:, B:, C:,...	:F0:, :F1:, :F2:, ...
Line printer	LST:	LST:
Teletype or CRT	TTY:	TTY:
High speed reader	HSR	

Examples:

*,TTY:=TEST	Assemble the source file TEST.MAC and list the program on the console. No object code is generated. Useful for error check.
*SMALL,TTY:=B:TEST	Assemble TEST.MAC (found on disk drive B), place the object file in SMALL.REL, and list the program on the console.

2.2.2 Switches

A switch is a letter that is appended to the command string, preceded by a slash. It specifies an optional task to be performed during assembly. More than one switch can be used, but each must be preceded by a slash. (With the TEKDOS operating system, switches are preceded by commas or spaces. See Appendix A.) All switches are optional. The available switches are:

Switch	Action
O	Octal listing
H	Hexadecimal listing (default)
R	Force generation of an object file
L	Force generation of a listing file
C	Force generation of a cross reference file

Z Assemble Z80 opcodes (default for Z80 operating systems)

I Assemble 8080 opcodes (default for 8080 operating systems)

P Each /P allocates an extra 256 bytes of stack space for use during assembly. Use /P if stack overflow errors occur during assembly. Otherwise, not needed.

M Initialize Block Data Areas. If the programmer wants the area that is defined by the DS (Define Space) speudo-op initialized to zeros, then the programmer should use the /M switch in the command line. Otherwise, the space is not guaranteed to contain zeros. That is, DS does not automatically initialize the space to zeros.

X Usually used to suppress the listing of false conditionals. The following paragraph describes the /X switch more completely but in very technical terms.

The presence or absence of /X in the command line sets the initial current mode and the initial value of the default for listing or suppressing lines in false conditional blocks. /X sets the current mode and initial value of default to not-to-list. No /X sets current mode and initial value of default to list. Current mode determines whether false conditionals will be listed or suppressed. The initial value of the default is used with the .TFCOND pseudo-op so that .TFCOND is independent of .SFCOND and .LFCOND. If the program contains .SFCOND or .LFCOND, /X has no effect after .SFCOND or .LFCOND is encountered until a .TFCOND is encountered in the file. So /X has an effect only when used with a file that contains no conditional listing pseudo-ops or when used with .TFCOND.

Examples:

*=TEST/L	Assemble TEST.MAC, place the object file in TEST.REL and a listing file in TEST.PRN. (With ISIS-II, the listing file is TEST.LST.)
*=TEST/L/O	Same as above, but listing file addresses will be in octal.
*LAST=TEST/C	Assemble TEST.MAC, place the object file in LAST.REL and cross reference file in TEST.CRF. (See Chapter 3.)

2.3 FORMAT OF MACRO-80 SOURCE FILES

Input source lines of up to 132 characters in length are acceptable.

MACRO-80 preserves lower case letters in quoted strings and comments. All symbols, opcodes and pseudo-opcodes typed in lower case will be converted to upper case.

If the source file includes line numbers from an editor, each byte of the line number must have the high bit on. Line numbers from Microsoft's EDIT-80 Editor are acceptable.

2.3.1 Statements

Source files input to MACRO-80 consist of statements of the form:

```
[label:[:]]  [operator]  [arguments]    [;comment]
```

With the exception of the ISIS assembler $ controls (see Section 2.11), it is not necessary that statements begin in column 1. Multiple blanks or tabs may be used to improve readability.

If a label is present, it is the first item in the statement and is immediately followed by a colon. If it is followed by two colons, it is declared as PUBLIC (see ENTRY/PUBLIC, Section 2.6.10). For exmple:

```
     FOO::    RET
```

is equivalent to

```
     PUBLIC   FOO
     FOO:     RET
```

The next item after the label, or the first item on the line if no label is present, is an operator. An operator may be an 8086 mnemonic, pseudo-op, macro call or expression. The evaluation order is as follows:

1. Macro call

2. Mnemonic/Pseudo operation

3. Expression

Instead of flagging an expression as an error, the assembler treats it as if it were a DB statement (see Section 2.6.4).

The arguments following the operator will, of course, vary in form according to the operator.

A comment always begins with a semicolon and ends with a carriage return. A comment may be a line by itself or it may be appended to a line that contains a statement. Extended comments can be entered using the .COMMENT pseudo operation (see Section 2.6.20).

2.3.2 Symbols

MACRO-80 symbols may be of any length, however, only the first six characters are significant. The following characters are legal in a symbol:

A-Z   0-9 $.   ?   @

With Microsoft's 8080/Z80/8086 assemblers, the underline character is also legal in a symbol. A symbol may not start with a digit. When a symbol is read, lower case is translated into upper case. If a symbol reference is followed by ## it is declared external (see also the EXT/EXTRN pseudo-op, Section 2.6.12).

2.3.3 Numeric Constants

The default base for numeric constants is decimal. This may be changed by the .RADIX pseudo-op (see Section 2.6.22). Any base from 2 (binary) to 16 (hexadecimal) may be selected. When the base is greater than 10, A-F are the digits following 9. If the first digit of the number is not numeric the number must be preceded by a zero.

Numbers are 16-bit unsigned quantities. A number is always evaluated in the current radix unless one of the following special notations is used:

```
  nnnnB     Binary
  nnnnD     Decimal
  nnnnO     Octal
  nnnnQ     Octal
  nnnnH     Hexadecimal
X'nnnn'     Hexadecimal
```

Overflow of a number beyond two bytes is ignored and the result is the low order 16-bits.

A character constant is a string comprised of zero, one or two ASCII characters, delimited by quotation marks, and used in a non-simple expression. For example, in the statement

```
        DB      'A' + 1
```

'A' is a character constant. But the statement

```
        DB      'A'
```

uses 'A' as a string because it is in a simple expression. The rules for character constant delimiters are the same as for strings.

A character constant comprised of one character has as its value the ASCII value of that character. That is, the high order byte of the value is zero, and the low order byte is the ASCII value of the character. For example, the value of the constant 'A' is 41H.

A character constant comprised of two characters has as its value the ASCII value of the first character in the high order byte and the ASCII value of the second character in the low order byte. For example, the value of the character constant "AB" is 41H*256+42H.

2.3.4 Strings

A string is comprised of zero or more characters delimited by quotation marks. Either single or double quotes may be used as string delimiters. The delimiter quotes may be used as characters if they appear twice for every character occurrence desired. For example, the statement

```
        DB      "I am ""great"" today"
```

stores the string

```
                I am "great" today
```

If there are zero characters between the delimiters, the string is a null string.

2.4 EXPRESSION EVALUATION

2.4.1 Arithmetic And Logical Operators

The following operators are allowed in expressions. The operators are listed in order of precedence.

NUL

LOW, HIGH

*, /, MOD, SHR, SHL

Unary Minus

+, -

EQ, NE, LT, LE, GT, GE

NOT

AND

OR, XOR

Parentheses are used to change the order of precedence. During evaluation of an expression, as soon as a new operator is encountered that has precedence less than or equal to the last operator encountered, all operations up to the new operator are performed. That is, subexpressions involving operators of higher precedence are computed first.

All operators except +, -, *, / must be separated from their operands by at least one space.

The byte isolation operators (HIGH, LOW) isolate the high or low order 8 bits of an Absolute 16-bit value. If a relocatable value is supplied as an operand, HIGH and LOW will treat it as if it were relative to location zero.

2.4.2 Modes

All symbols used as operands in expressions are in one of the following modes: Absolute, Data Relative, Program (Code) Relative or COMMON. (See Section 2.6 for the ASEG, CSEG, DSEG and COMMON pseudo-ops.) Symbols assembled under the ASEG, CSEG (default), or DSEG pseudo-ops are in Absolute, Code Relative or Data Relative mode respectively.

The number of COMMON modes in a program is determined by the number of COMMON blocks that have been named with the COMMON pseudo-op. Two COMMON symbols are not in the same mode unless they are in the same COMMON block. In any operation other than addition or subtraction, the mode of both operands must be Absolute.

If the operation is addition, the following rules apply:

1. At least one of the operands must be Absolute.

2. Absolute + <mode> = <mode>

If the operation is subtraction, the following rules apply:

1. <mode> - Absolute = <mode>

2. <mode> - <mode> = Absolute
   where the two <mode>s are the same.

Each intermediate step in the evaluation of an expression must conform to the above rules for modes, or an error will be generated. For example, if FOO, BAZ and ZAZ are three Program Relative symbols, the expression

```
FOO + BAZ - ZAZ
```

will generate an R error because the first step (FOO + BAZ) adds two relocatable values. (One of the values must be Absolute.) This problem can always be fixed by inserting parentheses. So that

```
FOO + (BAZ - ZAZ)
```

is legal because the first step (BAZ - ZAZ) generates an Absolute value that is then added to the Program Relative value, FOO.

2.4.3 Externals

Aside from its classification by mode, a symbol is either External or not External. (See EXT/EXTRN, Section 2.6.12.) An External value must be assembled into a two-byte field. (Single-byte Externals are not supported.) The following rules apply to the use of Externals in expressions:

1. Externals are legal only in addition and subtraction.

2. If an External symbol is used in an expression, the result of the expression is always External.

3. When the operation is addition, either operand (but not both) may be External.

4. When the operation is subtraction, only the first operand may be External.

2.5 OPCODES AS OPERANDS

8080 opcodes are valid one-byte operands. Note that only the first byte is a valid operand. For example:

```
MVI     A,(JMP)
ADI     (CPI)
MVI     B,(RNZ)
CPI     (INX H)
ACI     (LXI B)
MVI     C,MOV A,B
```

Errors will be generated if more than one byte is included in the operand -- such as (CPI 5), LXI B,LABEL1) or (JMP LABEL2).

Opcodes used as one-byte operands need not be enclosed in parentheses.

NOTE

Opcodes are not valid operands in Z80 mode.

2.6 PSEUDO OPERATIONS

2.6.1 ASEG

ASEG

ASEG sets the location counter to an absolute segment of memory. The location of the absolute counter will be that of the last ASEG (default is 0), unless an ORG is done after the ASEG to change the location. The effect of ASEG is also achieved by using the code segment (CSEG) pseudo operation and the /P switch in LINK-80. See also Section 2.6.28

2.6.2 COMMON

COMMON /<block name>/

COMMON sets the location counter to the selected common block in memory. The location is always the beginning of the area so that compatibility with the FORTRAN COMMON statement is maintained. If <block name> is omitted or consists of spaces, it is considered to be blank common. See also Section 2.6.28.

2.6.3 CSEG

CSEG

CSEG sets the location counter to the code relative segment of memory. The location will be that of the last CSEG (default is 0), unless an ORG is done after the CSEG to change the location. CSEG is the default condition of the assembler (the INTEL assembler defaults to ASEG). See also Section 2.6.28.

2.6.4 DB - Define Byte

DB <exp>[,<exp>...]

DB <string>[<string>...]

The arguments to DB are either expressions or strings. DB stores the values of the expressions or the characters of the strings in successive memory locations beginning with the current location counter.

Expressions must evaluate to one byte. (If the high byte of the result is 0 or 255, no error is given; otherwise, an A error results.)

Strings of three or more characters may not be used in expressions (i.e., they must be immediately followed by a comma or the end of the line). The characters in a string are stored in the order of appearance, each as a one-byte value with the high order bit set to zero.

Example:

```
  0000'    41 42          DB       'AB'
  0002'    42             DB       'AB' AND 0FFH
  0003'    41 42 43       DB       'ABC'
```

2.6.5 DC - Define Character

DC <string>

DC stores the characters in <string> in successive memory locations beginning with the current location counter. As with DB, characters are stored in order of appearance, each as a one-byte value with the high order bit set to zero. However, DC stores the last character of the string with the high order bit set to one. An error will result if the argument to DC is a null string.

2.6.6 DS - Define Space

DS <exp>

DS reserves an area of memory. The value of <exp> gives the number of bytes to be allocated. All names used in <exp> must be previously defined (i.e., all names known at that point on pass 1). Otherwise, a V error is generated during pass 1 and a U error may be generated during pass 2. If a U error is not generated during pass 2, a phase error will probably be generated because the DS generated no code on pass 1.

2.6.7 DSEG

DSEG

DSEG sets the location counter to the Data Relative segment of memory. The location of the data relative counter will be that of the last DSEG (default is 0), unless an ORG is done after the DSEG to change the location. See also Section 2.6.28.

2.6.8 DW - Define Word

```
DW      <exp>[,<exp>...]
```

DW stores the values of the expressions in successive memory locations beginning with the currenc location counter. Expressions are evaluated as 2-byte (word) values.

2.6.9 END

```
END     [<exp>]
```

The END statement specifies the end of the program. If <exp> is present, it is the start address of the program. If <exp> is not present, then no start address is passed to LINK-80 for that program.

NOTE

> If an assembly language program is the main program, a start address (label) must be specified. If not, LINK-80 will issue a "no start address" error. If the program is a subroutine to a FORTRAN program (for example), the start address is not required because FORTRAN has supplied one.

2.6.10 ENTRY/PUBLIC

```
ENTRY  <name>[,<name>...]
 or
PUBLIC <name>[,<name>...]
```

ENTRY or PUBLIC declares each name in the list as internal and therefore available for use by this program and other programs to be loaded concurrently. All of the names in the list must be defined in the current program or a U error results. An M error is generated if the name is an external name or common-blockname.

2.6.11 EQU

<name> EQU <exp>

EQU assigns the value of <exp> to <name>. If <exp> is external, an error is generated. If <name> already has a value other than <exp>, an M error is generated.

2.6.12 EXT/EXTRN

```
EXT    <name>[,<name>...]
 or
EXTRN  <name>[,<name>...]
```

EXT or EXTRN declares that the name(s) in the list are external (i.e., defined in a different program). If any item in the list references a name that is defined in the current program, an M error results. A reference to a name where the name is followed immediately by two pound signs (e.g., NAME##) also declares the name as external.

2.6.13 INCLUDE

INCLUDE <filename>

The INCLUDE pseudo-op applies only to CP/M versions of MACRO-80. The pseudo-ops INCLUDE, $INCLUDE and MACLIB are synonymous.

The INCLUDE pseudo-op assembles source statements from an alternate source file into the current source file. Use of INCLUDE eliminates the need to repeat an often-used sequence of statements in the current source file.

<filename> is any valid specification, as determined by the operating system. Defaults for filename extensions and device names are the same as those in a MACRO-80 command line.

The INCLUDE file is opened and assembled into the current source file immediately following the INCLUDE statement. When end-of-file is reached, assembly resumes with the statement following INCLUDE.

On a MACRO-80 listing, a plus sign is printed between the assembled code and the source line on each line assembled from an INCLUDE file. (See Section 2.12.)

Nested INCLUDEs are not allowed. If encountered, they will result in an objectionable syntax error 'O'.

The file specified in the operand field must exist. If the file is not found, the error 'V' (value error) is given, and the INCLUDE is ignored.

2.6.14 NAME

NAME ('modname')

NAME defines a name for the module. Only the first six characters are significant in a module name. A module name may also be defined with the TITLE pseudo-op. In the absence of both the NAME and TITLE pseudo-ops, the module name is created from the source file name.

2.6.15 ORG - Define Origin

ORG <exp>

The location counter is set to the value of <exp> and the assembler assigns generated code starting with that value. All names used in <exp> must be known on pass 1, and the value must either be absolute or in the same area as the location counter.

2.6.16 PAGE

PAGE [<exp>]

PAGE causes the assembler to start a new output page. The value of <exp>, if included, becomes the new page size (measured in lines per page) and must be in the range 10 to 255. The default page size is 50 lines per page. The assembler puts a form feed character in the listing file at the end of a page.

2.6.17 SET

<name> SET <exp>

SET is the same as EQU, except no error is generated if <name> is already defined.

2.6.18 SUBTTL

SUBTTL <text>

SUBTTL specifies a subtitle to be listed on the line after the title (see TITLE, Section 2.6.19) on each page heading. <text> is truncated after 60 characters. Any number of SUBTTLs may be given in a program.

2.6.19 TITLE

TITLE <text>

TITLE specifies a title to be listed on the first line of each page. If more than one TITLE is given, a Q error results. The first six characters of the title are used as the module name unless a NAME pseudo operation is used. If neither a NAME or TITLE pseudo-op is used, the module name is created from the source filename.

2.6.20 .COMMENT

.COMMENT <delim><text><delim>

The first non-blank character encountered after .COMMENT is the delimiter. The following <text> comprises a comment block which continues until the next occurrence of <delimiter> is encountered. For example, using an asterisk as the delimiter, the format of the comment block would be:

```
.COMMENT  *
any amount of text entered
here as the comment block
    .
    .
    .     *
;return to normal mode
```

2.6.21 .PRINTX

.PRINTX <delim><text><delim>

The first non-blank character encountered after .PRINTX is the delimiter. The following text is listed on the terminal during assembly until another occurrence of the delimiter is encountered. .PRINTX is useful for displaying progress through a long assembly or for displaying the value of conditional assembly switches. For example:

```
IF      CPM
.PRINTX /CPM version/
ENDIF
```

NOTE

.PRINTX will output on both passes. If only one printout is desired, use the IF1 or IF2 pseudo-op. For example:

```
IF2
IF CPM
.PRINTX /CPM version/
ENDIF
ENDIF
```

will only print if CPM is true and M80 is in pass 2.

2.6.22 .RADIX

.RADIX <exp>

The default base (or radix) for all constants is decimal. The .RADIX statement allows the default radix to be changed to any base in the range 2 to 16. For example:

```
MOVI    BX,0FFH
.RADIX 16
MOVI    BX,0FF
```

The two MOVIs in the example are identical. The <exp> in a .RADIX statement is always in decimal radix, regardless of the current radix.

2.6.23 .Z80

.Z80 enables the assembler to accept Z80 opcodes. This is the default condition when the assembler is running on a Z80 operating system. Z80 mode may also be set by appending the Z switch to the MACRO-80 command string -- see Section 2.2.2.

2.6.24 .8080

.8080 enables the assembler to accept 8080 opcodes. This is the default condition when the assembler is running on an 8080 operating system. 8080 mode may also be set by appending the I switch to the MACRO-80 command string -- see Section 2.2.2.

2.6.25 .REQUEST

```
.REQUEST <filename>[,<filename>...]
```

.REQUEST sends a request to the LINK-80 loader to search the filenames in the list for undefined globals. The filenames in the list should be in the form of legal symbols. They should not include filename extensions or disk specifications. LINK-80 supplies a default extension and assumes the default disk drive.

2.6.26 Conditional Pseudo Operations

The conditional pseudo operations are:

IF/IFT <exp>	True if <exp> is not 0.
IFE/IFF <exp>	True if <exp> is 0.
IF1	True if pass 1.
IF2	True if pass 2.
IFDEF <symbol>	True if <symbol> is defined or has been declared External.
IFNDEF <symbol>	True if <symbol> is undefined or not declared External.
IFB <arg>	True if <arg> is blank. The angle brackets around <arg> are required.
IFNB <arg>	True if <arg> is not blank. Used for testing when dummy parameters are supplied. The angle brackets around <arg> are required.
IFIDN <arg1>,<arg2>	True if the string <arg1> is IDeNtical to the string <arg2>. The angle brackets around <arg1> and <arg2> are required.
IFDIF <arg1>,<arg2>	True if the string <arg1> is DIFferent from the string <arg2>. The angle brackets around <arg1> and <arg2> are required.

All conditionals use the following format:

```
IFxx   [argument]
  .
  .
  .
[ELSE
  .
  .
  .     ]
ENDIF
```

Conditionals may be nested to any level. Any argument to a conditional must be known on pass 1 to avoid V errors and incorrect evaluation. For IF, IFT, IFF, and IFE the expression must involve values which were previously defined and the expression must be absolute. If the name is defined after an IFDEF or IFNDEF, pass 1 considers the name to be undefined, but it will be defined on pass 2.

2.6.26.1 ELSE - Each conditional pseudo operation may optionally be used with the ELSE pseudo operation which allows alternate code to be generated when the opposite condition exists. Only one ELSE is permitted for a given IF, and an ELSE is always bound to the most recent, open IF. A conditional with more than one ELSE or an ELSE without a conditional will cause a C error.

2.6.26.2 ENDIF - Each IF must have a matching ENDIF to terminate the conditional. Otherwise, an 'Unterminated conditional' message is generated at the end of each pass. An ENDIF without a matching IF causes a C error.

2.6.27 Listing Control Pseudo Operations

Output to the listing file can be controlled by two pseudo-ops:

    .LIST   and   .XLIST

If a listing is not being made, these pseudo-ops have no effect. .LIST is the default condition. When a .XLIST is encountered, source and object code will not be listed until a .LIST is encountered.

The output of false conditional blocks is controlled by three pseudo-ops: .SFCOND, .LFCOND, and .TFCOND.

These pseudo-ops give the programmer control over four cases.

1. Normally list false conditionals
   For this case, the programmer simply allows the default mode to control the listing. The default mode is list false conditionals. If the programmer decides to suppress false conditionals, the /X switch can be issued in the command line instead of editing the source file.

2. Normally suppress false conditionals
   For this case, the programmer issues the .TFCOND pseudo-op in the program file. .TFCOND reverses (toggles) the default, causing false conditionals to be suppressed. If the programmer decides to list false conditionals, the /X switch can be issued in the command line instead of editing the source file.

3. Always suppress/list false conditionals
   For these cases, the programmer issues either the .SFCOND pseudo-op to always suppress false conditionals, or the .LFCOND pseudo-op to always list all false conditionals.

4. Suppress/list some false conditionals
   For this case, the programmer has decided for most false conditionals whether to list or suppress, but for some false conditionals the programmer has not yet decided. For the false conditionals decided about, use .SFCOND or .LFCOND. For those not yet decided, use .TFCOND. .TFCOND sets the current and default settings to the opposite of the default. Initially, the default is set by giving /X or no /X in the command line. Two subcases exist:

   1. The programmer wants some false conditionals not to list unless /X is given. The programmer uses the .SFCOND and .LFCOND pseudo-ops to control which areas always suppress or list false conditionals. To selectively suppress some false conditionals, the programmer issues .TFCOND at the beginning of the conditional block and again at the end of the conditional block. (NOTE: The second .TFCOND should be issued so that the default setting will be the same as the initial setting. Leaving the default equal to the initial setting makes it easier to keep track of the default mode if there are many such areas.) If the conditional block evaluates as false, the lines will be suppressed. In this subcase, issuing the /X switch in the command line causes the conditional block affected by .TFCOND to list even if it evaluates as false.

2. The programmer wants some false conditionals to list unless /X is given. Two consecutive .TFCONDs places the conditional listing setting in initial state which is determined by the presence or absence of the /X switch in the command line (the first .TFCOND sets the default to not initial; the second to initial). The selected conditional block then responds to the /X switch: if a /X switch is issued in the command line, the conditional block is suppressed if false; if no /X switch is issued in the command line, the conditional block is listed even if false.

   The programmer then must reissue the .SFCOND or .LFCOND conditional listing pseudo-op to restore the suppress or list mode. Simply issuing another .TFCOND will not restore the prior mode, but will toggle the default setting. Since in this subcase, the next area of code is supposed to list or suppress false conditionals always, the programmer must issue .SFCOND or .LFCOND.

The three conditional listing pseudo-ops are summarized below.

PSEUDO-OP	DEFINITION
.SFCOND	Suppresses the listing of conditional blocks that evaluate as false.
.LFCOND	Restores the listing of conditional blocks that evaluate as false.
.TFCOND	Toggles the current setting which controls the listing false conditionals. .TFCOND sets the current and default setting to not default. If a /X switch is given in the MACRO-80 run command line for a file which contains .TFCOND, /X reverses the effect of .TFCOND.

The following chart illustrates the effects of the three pseudo-ops when encountered under /X and under no /X.

PSEUDO-OP	NO /X	/X
(none)	ON	OFF
.	.	.
.	.	.
.	.	.
.SFCOND	OFF	OFF
.	.	.
.	.	.
.	.	.
.LFCOND	ON	ON
.	.	.
.	.	.
.	.	.
.TFCOND	OFF	ON
.	.	.
.	.	.
.	.	.
.TFCOND	ON	OFF
.	.	.
.	.	.
.	.	.
.SFCOND	OFF	OFF
.	.	.
.	.	.
.	.	.
.TFCOND	OFF	ON
.TFCOND	ON	OFF
.	.	.
.	.	.
.	.	.
.TFCOND	OFF	ON

The output of cross reference information is controlled by .CREF and .XCREF. If the cross reference facility (see Chapter 3) has not been invoked, .CREF and .XCREF have no effect. The default condition is .CREF. When a .XCREF is encountered, no cross reference information is output until .CREF is encountered.

The output of MACRO/REPT/IRP/IRPC expansions is controlled by three pseudo-ops: .LALL, .SALL, and .XALL. .LALLlists the complete macro text for all expansions. .SALL suppresses lsiting of all text and object code produced by macros. .XALL is the default condition; a source line is listed only if it generates object code.

2.6.28 Relocation Pseudo Operations

The ability to create relocatable modules is one of the major features of Microsoft assemblers. Relocatable modules offer the advantages of easier coding and faster testing, debugging and modifying. In addition, it is possible to specify segments of assembled code that will later be loaded into RAM (the Data Relative segment) and ROM/PROM (the Code Relative segment). The pseudo operations that select relocatable areas are CSEG and DSEG. The ASEG pseudo-op is used to generate non-relocatable (absolute) code. The COMMON pseudo-op creates a common data area for every COMMON block that is named in the program.

The default mode for the assembler is Code Relative. That is, assembly begins with a CSEG automatically executed and the location counter in the Code Relative mode, pointing to location 0 in the Code Relative segment of memory. All subsequent instructions will be assembled into the Code Relative segment of memory until an ASEG or DSEG or COMMON pseudo-op is executed. For example, the first DSEG encountered sets the location counter to location zero in the Data Relative segment of memory. The following code is assembled in the Data Relative mode, that is, it is assigned to the Data Relative segment of memory. If a subsequent CSEG is encountered, the location counter will return to the next free location in the Code Relative segment and so on.

The ASEG, DSEG, CSEG pseudo-ops never have operands. If you wish to alter the current value of the location counter, use the ORG pseudo-op.

2.6.28.1 ORG Pseudo-op - At any time, the value of the location counter may be changed by use of the the ORG pseudo-op. The form of the ORG statement is:

```
ORG     <exp>
```

where the value of <exp> will be the new value of the location counter in the current mode. All names used in <exp> must be known on pass 1 and the value of <exp> must be either Absolute or in the current mode of the location counter. For example, the statements

```
DSEG
ORG     50
```

set the Data Relative location counter to 50, relative to the start of the Data Relative segment of memory.

2.6.28.2 LINK-80 - The LINK-80 linking loader (see Chapter 4 of this manual) combines the segments and creates each relocatable module in memory when the program is loaded. The origins of the relocatable segments are not fixed until the program is loaded and the origins are assigned by LINK-80. The command to LINK-80 may contain user-specified origins through the use of the /P (for Code Relative) and /D (for Data and COMMON segments) switches.

For example, a program that begins with the statements

```
                              ASEG
                              ORG       800H
```

and is assembled entirely in Absolute mode will always load beginning at 800 unless the ORG statement is changed in the source file. However, the same program, assembled in Code Relative mode with no ORG statement, may be loaded at any specified address by appending the /P:<address> switch to the LINK-80 command string.

2.6.29 Relocation Before Loading

Two pseudo-ops, .PHASE and .DEPHASE, allow code to be located in one area, but executed only at a different, specified area.

For example:

```
 0000'                          .PHASE   100H
 0100     E8 0003     FOO:      CALL     BAZ
 0103     E9 FF01               JMP      ZOO
 0106     C3          BAZ:      RET
                                .DEPHASE
 0007'    E9 FFFB     ZOO:      JMP      5
```

All labels within a .PHASE block are defined as the absolute value from the origin of the phase area. The code, however, is loaded in the current area (i.e., from 0' in this example). The code within the block can later be moved to 100H and executed.

2.7 MACROS AND BLOCK PSEUDO OPERATIONS

The macro facilities provided by MACRO-80 include three repeat pseudo operations: repeat (REPT), indefinite repeat (IRP), and indefinite repeat character (IRPC). A macro definition operation (MACRO) is also provided. Each of these four macro operations is terminated by the ENDM pseudo operation.

2.7.1 Terms

For the purposes of discussion of macros and block operations, the following terms will be used:

1. <dummy> is used to represent a dummy parameter. All dummy parameters are legal symbols that appear in the body of a macro expansion.

2. <dummylist> is a list of <dummy>s separated by commas.

3. <arglist> is a list of arguments separated by commas. <arglist> must be delimited by angle brackets. Two angle brackets with no intervening characters (<>) or two commas with no intervening characters enter a null argument in the list. Otherwise an argument is a character or series of characters terminated by a comma or >. With angle brackets that are nested inside an <arglist>, one level of brackets is removed each time the bracketed argument is used in an <arglist>. See example, Section 2.7.5.) A quoted string is an acceptable argument and is passed as such. Unless enclosed in brackets or a quoted string, leading and trailing spaces are deleted from arguments.

4. <paramlist> is used to represent a list of actual parameters separated by commas. No delimiters are required (the list is terminated by the end of line or a comment), but the rules for entering null parameters and nesting brackets are the same as described for <arglist>. (See example, Section 2.7.5)

2.7.2 REPT-ENDM

```
REPT <exp>
 .
 .
 .
ENDM
```

The block of statements between REPT and ENDM is repeated <exp> times. <exp> is evaluated as a 16-bit unsigned number. If <exp> contains any external or undefined terms, an error is generated. Example:

```
SET 0
REPT 10      ;generates DB 1 - DB 10
SET X+1
DB X
ENDM
```

2.7.3 IRP-ENDM

```
IRP <dummy>,<arglist>
 .
 .
 .
ENDM
```

The <arglist> must be enclosed in angle brackets. The number of arguments in the <arglist> determines the number of times the block of statements is repeated. Each repetition substitutes the next item in the <arglist> for every occurrence of <dummy> in the block. If the <arglist> is null (i.e., <>), the block is processed once with each occurrence of <dummy> removed. For example:

```
IRP X,<1,2,3,4,5,6,7,8,9,10>
DB X
ENDM
```

generates the same bytes as the REPT example.

2.7.4 IRPC-ENDM

```
IRPC <dummy>,string (or <string>)
 .
 .
 .
ENDM
```

IRPC is similar to IRP but the arglist is replaced by a string of text and the angle brackets around the string are optional. The statements in the block are repeated once for each character in the string. Each repetition substitutes the next character in the string for every occurrence of <dummy> in the block. For example:

```
IRPC X,0123456789
DB X+1
ENDM
```

generates the same code as the two previous examples.

2.7.5 MACRO

Often it is convenient to be able to generate a given sequence of statements from various places in a program, even though different parameters may be required each time the sequence is used. This capability is provided by the MACRO statement.
The form is

```
<name> MACRO <dummylist>
         .
         .
         .
        ENDM
```

where <name> conforms to the rules for forming symbols. <name> is the name that will be used to invoke the macro. The <dummy>s in <dummylist> are the parameters that will be changed (replaced) each time the MACRO is invoked. The statements before the ENDM comprise the body of the macro. During assembly, the macro is expanded every time it is invoked but, unlike REPT/IRP/IRPC, the macro is not expanded when it is encountered.

The form of a macro call is

```
<name> <paramlist>
```

where <name> is the name supplied in the MACRO definition, and the parameters in <paramlist> will replace the <dummy>s in the MACRO <dummylist> on a one-to-one basis. The number of items in <dummylist> and <paramlist> is limited only by the length of a line. The number of parameters used when the macro is called need not be the same as the number of <dummy>s in <dummylist>. If there are more parameters than <dummmy>s, the extras are ignored. If there are fewer, the extra <dummy>s will be made null. The assembled code will contain the macro expansion code after each macro call.

NOTE

A dummy parameter in a MACRO/REPT/IRP/IRPC is always recognized exclusively as a dummmy parameter. Register names such as A and B will be changed in the expansion if they were used as dummy parameters.

Here is an example of a MACRO definition that defines a macro called FOO:

```
        FOO     MACRO   X
        Y       SET     0
                REPT    X
        Y       SET     Y+1
                DB      Y
                ENDM
                ENDM
```

This macro generates the same code as the previous three examples when the call

```
                FOO     10
```

is executed.

Another example, which generates the same code, illustrates the removal of one level of brackets when an argument is used as an arglist:

```
        FOO     MACRO   X
                IRP     Y,<X>
                DB      Y
                ENDM
                ENDM
```

When the call

```
                FOO     <1,2,3,4,5,6,7,8,9,10>
```

is made, the macro expansion looks like this:

```
                IRP     Y,<1,2,3,4,5,6,7,8,9,10>
                DB      Y
                ENDM
```

2.7.6 ENDM

Every REPT, IRP, IRPC and MACRO pseudo-op must be terminated with the ENDM pseudo-op. Otherwise, the 'Unterminated REPT/IRP/IRPC/MACRO' message is generated at the end of each pass. An unmatched ENDM causes an O error.

2.7.7 EXITM

The EXITM pseudo-op is used to terminate a REPT/IRP/IRPC or MACRO call. When an EXITM is executed, the expansion is exited immediately and any remaining expansion or repetition is not generated. If the block containing the EXITM is nested within another block, the outer level continues to be expanded.

2.7.8 LOCAL

LOCAL <dummylist>

The LOCAL pseudo-op is allowed only inside a MACRO definition. When LOCAL is executed, the assembler creates a unique symbol for each <dummy> in <dummylist> and substitutes that symbol for each occurrence of the <dummy> in the expansion. These unique symbols are usually used to define a label within a macro, thus eliminating multiply-defined labels on successive expansions of the macro. The symbols created by the assembler range from ..0001 to ..FFFF. Users will therefore want to avoid the form ..nnnn for their own symbols. If LOCAL statements are used, they must be the first statements in the macro definition.

2.7.9 Special Macro Operators And Forms

& The ampersand is used in a macro expansion to concatenate text or symbols. A dummy parameter that is in a quoted string will not be substituted in the expansion unless it is immediately preceded by &. To form a symbol from text and a dummy, put & between them. For example:

```
ERRGEN   MACRO    X
ERROR&X:PUSH     BX
         MOVI     BX,'&X'
         JMP      ERROR
         ENDM
```

In this example, the call ERRGEN A will generate:

```
ERRORA: PUSH    B
   MOVI     BX,'A'
   JMP      ERROR
```

;; In a block operation, a comment preceded by two semicolons is not saved as part of the expansion (i.e., it will not appear on the listing even under .LALL). A comment preceded by one semicolon, however, will be preserved and appear in the expansion.

! When an exclamation point is used in an argument, the next character is entered literally (i.e., !; and <;> are equivalent).

NUL NUL is an operator that returns true if its argument (a parameter) is null. The remainder of a line after NUL is considered to be the argument to NUL. The conditional

IF NUL argument

is false if, during the expansion, the first character of the argument is anything other than a semicolon or carriage return. It is recommended that testing for null parameters be done using the IFB and IFNB conditionals.

% The percent sign is used only in a macro argument. % converts the expression that follows it (usually a symbol) to a number in the current radix. During macro expansion, the number derived from converting the expression is substituted for the dummy. Using the % special operator allows a macro call by value. (Usually, a macro call is a call by reference with the text of the macro argument substituting exactly for the dummy.)

The expression following the % must conform to the same rules as the DS (Define Space) pseudo-op. A valid expression returning a non-relocatable constant is required.

EXAMPLE: Normally, LB, the argument to MAKLAB, would be substituted for Y, the argument to MACRO, as a string. The % causes LB to be converted to a non-relocatable constant which is then substituted for Y. Without the % special operator, the result of assembly would be 'Error LB' rather than 'Error 1', etc.

```
MAKLAB    MACRO     Y
ERR&Y:    DB        'Error &Y',0
          ENDM
MAKERR    MACRO     X
LB        SET       0
          REPT      X
LB        SET       LB+1
          MAKLAB    %LB
          ENDM
          ENDM
```

When called by MAKERR 3, the assembler will generate:

```
ERR1:  DB     'Error 1',0
ERR2:  DB     'Error 2',0
ERR3:  DB     'Error 3',0
```

TYPE The TYPE operator returns a byte that describes two characteristics of its argument: 1) the mode, and 2) whether it is External or not. The argument to TYPE may be any expression (string, numeric, logical). If the expression is invalid, TYPE returns zero.

The byte that is returned is configured as follows:

The lower two bits are the mode. If the lower two bits are:

0	the mode is Absolute
1	the mode is Program Relative
2	the mode is Data Relative
3	the mode is Common Relative

The high bit (80H) is the External bit. If the high bit is on, the expression contains an External. If the high bit is off, the expression is local (not External).

The Defined bit is 20H. This bit is on if the expression is locally defined, and it is off if the expression is undefined or external. If neither bit is on, the expression is invalid.

TYPE is usually used inside macros, where an argument type may need to be tested to make a decision regarding program flow. For example:

```
FOO     MACRO    X
        LOCAL    Z
Z       SET TYPE X
IF      Z...
```

2.8 USING Z80 PSEUDO-OPS

When using the MACRO-80 assembler, the following Z80 pseudo-ops are valid. The function of each pseudo-op is equivalent to that of its counterpart.

Z80 pseudo-op	Equivalent pseudo-op
COND	IFT
ENDC	ENDIF
*EJECT	PAGE
DEFB	DB
DEFS	DS
DEFW	DW
DEFM	DB
DEFL	SET
GLOBAL	PUBLIC
EXTERNAL	EXTRN

The formats, where different, conform to the previous format. That is, DEFB and DEFW are permitted a list of arguments (as are DB and DW), and DEFM is permitted a string or numeric argument (as is DB).

2.9 SAMPLE ASSEMBLY

```
A>M80

*EXMPL1,TTY:=EXMPL1

        MAC80 3.2        PAGE    1


                            00100    ;CSL3(P1,P2)
                            00200    ;SHIFT P1 LEFT CIRCULARLY 3 BITS
                            00300    ;RETURN RESULT IN P2
                            00400            ENTRY   CSL3
                            00450    ;GET VALUE OF FIRST PARAMETER
                            00500    CSL3:
  0000'    7E               00600            MOV     A,M
  0001'    23               00700            INX     H
  0002'    66               00800            MOV     H,M
  0003'    6F               00900            MOV     L,A
                            01000    ;SHIFT COUNT
  0004'    06 03            01100            MVI     B,3
  0006'    AF               01200    LOOP:   XRA     A
                            01300    ;SHIFT LEFT
  0007'    29               01400            DAD     H
                            01500    ;ROTATE IN CY BIT
  0008'    17               01600            RAL
  0009'    85               01700            ADD     L
  000A'    6F               01800            MOV     L,A
                            01900    ;DECREMENT COUNT
  000B'    05               02000            DCR     B
                            02100    ;ONE MORE TIME
  000C'    C2 0006'         02200            JNZ     LOOP
  000F'    EB               02300            XCHG
                            02400    ;SAVE RESULT IN SECOND PARAMETER
  0010'    73               02500            MOV     M,E
  0011'    23               02600            INX     H
  0012'    72               02700            MOV     M,D
  0013'    C9               02800            RET
                            02900            END


        MAC80 3.2        PAGE    S


CSL3    0000I'  LOOP    0006'


No  Fatal error(s)
```

2.10 MACRO-80 ERRORS

MACRO-80 errors are indicated by a one-character flag in column one of the listing file. If a listing file is not being printed on the terminal, each erroneous line is also printed or displayed on the terminal. Below is a list of the MACRO-80 Error Codes:

A Argument error
Argument to pseudo-op is not in correct format or is out of range (.PAGE 1; .RADIX 1; PUBLIC 1; JMPS TOOFAR).

C Conditional nesting error
ELSE without IF, ENDIF without IF, two ELSEs on one IF.

D Double Defined symbol
Reference to a symbol which is multiply defined.

E External error
Use of an external illegal in context (e.g., FOO SET NAME##; MOVI AX,2-NAME##).

M Multiply Defined symbol
Definition of a symbol which is multiply defined.

N Number error
Error in a number, usually a bad digit (e.g., 8Q).

O Bad opcode or objectionable syntax
ENDM, LOCAL outside a block; SET, EQU or MACRO without a name; bad syntax in an opcode; or bad syntax in an expression (mismatched parenthesis, quotes, consecutive operators, etc.).

P Phase error
Value of a label or EQU name is different on pass 2.

Q Questionable
Usually means a line is not terminated properly. This is a warning error (e.g. MOV AX,BX,).

R Relocation
Illegal use of relocation in expression, such as abs-rel. Data, code and COMMON areas are relocatable.

U Undefined symbol
A symbol referenced in an expression is not defined. (For certain pseudo-ops, a V error is printed on pass 1 and a U on pass 2.)

V Value error
On pass 1 a pseudo-op which must have its value known on pass 1 (e.g., .RADIX, .PAGE, DS, IF, IFE, etc.), has a value which is undefined. If the symbol is defined later in the program, a U error will not appear on the pass 2 listing.

Error Messages:

'No end statement encountered on input file'
No END statement: either it is missing or it is not parsed due to being in a false conditional, unterminated IRP/IRPC/REPT block or terminated macro.

'Unterminated conditional'
At least one conditional is unterminated at the end of the file.

'Unterminated REPT/IRP/IRPC/MACRO'
At least one block is unterminated.

[xx] [No] Fatal error(s) [,xx warnings]
The number of fatal errors and warnings. The message is listed on the CRT and in the list file.

2.11 COMPATIBILITY WITH OTHER ASSEMBLERS

The $EJECT and $TITLE controls are provided for compatability with INTEL's ISIS assembler. The dollar sign must appear in column 1 only if spaces or tabs separate the dollar sign from the control word. The control

```
$EJECT
```

is the same as the MACRO-80 PAGE pseudo-op.
The control

```
$TITLE('text')
```

is the same as the MACRO-80 SUBTTL <text> pseudo-op.

The INTEL operands PAGE and INPAGE generate Q errors when used with the MACRO-80 CSEG or DSEG pseudo-ops. These errors are warnings; the assembler ignores the operands.

When MACRO-80 is entered, the default for the origin is Code Relative 0.

With the INTEL ISIS assembler, the default is Absolute 0.

With MACRO-80, the dollar sign ($) is a defined constant that indicates the value of the location counter at the start of the statement. Other assemblers may use a decimal point or an asterisk. Other constants are defined by MACRO-80 to have the following values:

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

2.12 FORMAT OF LISTINGS

On each page of a MACRO-80 listing, the first two lines have the form:

```
[TITLE text]    M80 3.3     PAGE x[-y]
[SUBTTL text]
```

where:

1. TITLE text is the text supplied with the TITLE pseudo-op, if one was given in the source program.

2. x is the major page number, which is incremented only when a form feed is encountered in the source file. (When using Microsoft's EDIT-80 text editor, a form feed is inserted whenever a page mark is done.) When the symbol table is being printed, x = S.

3. y is the minor page number, which is incremented whenever the .PAGE pseudo-op is encountered in the source file, or whenever the current page size has been filled.

4. SUBTTL text is the text supplied with the SUBTTL pseudo-op, if one was given in the source program.

Next, a blank line is printed, followed by the first line of output.

A line of output on a MACRO-80 listing has the following form:

```
[crf#]     [error] loc#m    |xx | xxxx|...    source
```

If cross reference information is being output, the first item on the line is the cross reference number, followed by a tab.

A one-letter error code followed by a space appears next on the line, if the line contains an error. If there is no error, a space is printed. If there is no cross reference number, the error code column is the first column on the listing.

The value of the location counter appears next on the line. It is a 4-digit hexadecimal number or 6-digit octal number, depending on whether the /O or /H switch was given in the MACRO-80 command string.

The character at the end of the location counter value is the mode indicator. It will be one of the following symbols:

'	Code Relative
"	Data Relative
!	COMMON Relative
<space>	Absolute
*	External

Next, three spaces are printed followed by the assembled code. One-byte values are followed by a space. Two-byte values are followed by a mode indicator. Two-byte values are printed in the opposite order they are stored in, i.e., the high order byte is printed first. Externals are either the offset or the value of the pointer to the next External in the chain.

If a line of output on a MACRO-80 listing is from an INCLUDE file, the character 'C' is printed after the assembled code on that line. If a line of output is part of a text expansion (MACRO, REPT, IRP, IRPC) a plus sign '+' is printed after the assembled code on that line.

The remainder of the line contains the line of source code, as it was input.

Example:

```
0C49 3A A91Z' C+ LDA LCOUNT
```

'C+' indicates this line is from an INCLUDE file and part of a macro expansion.

2.12.1 Symbol Table Listing

In the symbol table listing, all the macro names in the program are listed alphabetically, followed by all the symbols in the program, listed alphabetically. After each symbol, a tab is printed, followed by the value of the symbol. If the symbol is Public, an I is printed immediately after the value. The next character printed will be one of the following:

U	Undefined symbol.
C	COMMON block name. (The "value" of the COMMON block is its length (number of bytes) in hexadecimal or octal.)
*	External symbol.
<space>	Absolute value.
'	Program Relative value.
"	Data Relative value.
!	COMMON Relative value.

CHAPTER 4

LINK-80 LINKING LOADER

NOTE

If you are using the TEKDOS operating system, see Appendix A for proper command formats.

4.1 RUNNING LINK-80

The command to run LINK-80 is

```
L80
```

LINK-80 returns the prompt "*", indicating it is ready to accept commands.

4.2 COMMAND FORMAT

Each command to LINK-80 consists of a string of object filenames separated by commas. These are the files to be loaded by LINK-80. The command format is:

```
objfile1,objfile2,...objfilen
```

The default extension for all filenames is REL. Command lines are supported, that is, the invocation and command may be typed on the same line.

Example:

```
L80 MYPROG,YRPROG
```

Any filename in the LINK-80 command string can also specify a device name. The default device name with the CP/M operating system is the currently logged disk. The default device with the ISIS-II operating system is disk drive 0. The format is:

```
devl:objfilel,dev2:objfile2,...devn:objfilen
```

The device names are as listed in Section 2.2.1.

Example:

```
L80 MYPROG,A:YRPROG
```

After each line is typed, LINK-80 will load the specified files. After LINK finishes this process, it will list all symbols that remained undefined followed by an asterisk.

Example:

```
*MAIN

 DATA    0100    0200

 SUBRl*      (SUBRl is undefined)

*SUBRl

 DATA    0100    0300

*
```

Typically, to execute a MACRO-80 program and subroutines, the user types the list of filenames followed by /G (begin execution). To resolve any external, undefined symbols, you can first search your library routines (See Chapter 5, LIB-80) by appending the filenames, followed by /S, to the loader command string.

*MYLIB/S	Searches MYLIB.REL for unresolved global symbols
*/G	Starts execution

4.2.1 LINK-80 Switches

A number of switches may be given in the LINK-80 command string to specify actions affecting the loading or execution of the program(s). Each switch must be preceded by a slash (/). (With the TEKDOS operating system, switches are preceded by hyphens . See Appendix A.)

Switches may be placed wherever applicable in the command string:

1. At command level. It is possible for a switch to be the entire LINK-80 command, or to appear first in the command string. For example:

*/G	Tells LINK-80 to begin execution of program(s) already loaded
*/M	List all global references from program(s) already loaded
*/P:200,FOO	Load FOO, with program area beginning at address 200

2. Immediately after a filename. An S or N switch may refer to only one filename in the command string. Therefore, when the S or N switch is required, it is placed immediately after that filename, regardless of where the filename appears in the command string. For example:

   *MYLIB/S,MYPROG
   Search MYLIB.REL and load necessary library modules, then load MYPROG.REL.

   *MYPROG/N,MYPROG/E
   Load MYPROG.REL, save MYPROG.COM on disk and exit LINK-80.

3. At the end of the command string. Any required switches that affect the entire load process may be appended at the end of the command string. For example:

   *MYPROG/N,MYPROG/M/E
   Open a CP/M COM file called MYPROG.COM, load MYPROG.REL and list all global references. Exit LINK-80 and save the COM file.

   MYLIB/S,MYSUB,MYPROG/N,MYPROG/M/G
   Search MYLIB.REL, load and link MYSUB.REL and MYPROG.REL, open a CP/M COM file called MYPROG.COM, list all global references, save the COM file, and execute MYPROG.

The available switches are:

Switch	Action
R	Reset. Put loader back in its initial state. Use /R if you loaded the wrong file by mistake and want to restart. /R takes effect as soon as it is encountered in a command string.
E or E:Name	Exit LINK-80 and return to the operating system. The system library will be searched on the current disk to satisfy any existing undefined globals. Before exiting, LINK-80 prints three numbers: the start address, the address of the next available byte, and the number of 256-byte pages used. The optional form E:Name (where Name is a global symbol previously defined in one of the modules) uses Name for the start address of the program. Use /E to load a program and exit back to the monitor.
G or G:Name	Start execution of the program as soon as the current command line has been interpreted. The system library will be searched on the current disk to satisfy any existing undefined globals if they exist. Before execution actually begins, LINK-80 prints three numbers and a BEGIN EXECUTION message. The three numbers are the start address, the address of the next available byte, and the number of 256-byte pages used. The optional form G:Name (where Name is a global symbol previously defined in one of the modules) uses Name for the start address of the program.
N	If a <filename>/N is specified, the program will be saved on disk under the selected name (with a default extension of .COM for CP/M) when a /E or /G is done. A jump to the start of the program is inserted if needed so the program can run properly (at 100H for CP/M).

P and D

/P and /D allow the origin(s) to be set for the next program loaded. /P and /D take effect when seen (not deferred), and they have no effect on programs already loaded. The form is /P:<address> or /D:<address>, where <address> is the desired origin in the current typeout radix. (Default radix is hex. /O sets radix to octal; /H to hex.) LINK-80 does a default /P:<link origin>+3 (i.e., 103H for CP/M and 4003H for ISIS) to leave room for the jump to the start address. NOTE: Do not use /P or /D to load programs or data into the locations of the loader's jump to the start address (100H to 102H for CP/M) unless it is to load the start of the program there. If programs or data are loaded into these locations, the jump will not be generated.

If no /D is given, data areas are loaded before program areas for each module. If a /D is given, all Data and Common areas are loaded starting at the data origin and the program area at the program origin. Example:

```
*/P:200,FOO
Data     200     300
*/R
*/P:200 /D:400,FOO
Data     400     480
Program 200     280
```

U

List the origin and end of the program and data area and all undefined globals as soon as the current command line has been interpreted. The program information is only printed if a /D has been done. Otherwise, the program is stored in the data area.

M

List the origin and end of the program and data area, all defined globals and their values, and all undefined globals followed by an asterisk. The program information is only printed if a /D has been done. Otherwise, the program is stored in the data area.

S

Search the filename immediately preceding the /S in the command string to satisfy any undefined globals.

4.2.2 CP/M LINK-80 Switches

The following switches apply to CP/M versions only.

X — If a filename/N was specified, /X will cause the file to be saved in Intel ASCII HEX format with an extension of HEX.

Example: FOO/N/X/E will create an Intel ASCII HEX formatted load module named FOO.HEX.

Y — If a filename/N was specified, /Y will create a filename.SYM file when /E is entered. This file contains the names and addresses of all Globals for use with Digital Research's Symbolic Debugger, SID and ZSID.

Example: FOO/N/Y/E creates FOO.COM and FOO.SYM. MYPROG/N/X/Y/E creates MYPROG.HEX and MYPROG.SYM.

4.2.3 Sample Links

LINK AND GO

```
     A>L80
     *EXAMPL,EXMPL1/G
     DATA     3000     30AC
     [304F    30AC     49]
[BEGIN EXECUTION]

               1792           14336
              14336          -16383
             -16383              14
                 14             112
                112             896
     A>
```

LINK AND SAVE

```
     A>L80
     *EXAMPL,EXAMPL1,EXAM/N/E
     DATA     3000     30AC
     [304F     30AC     49]
     A>
```

Loads and links EXAMPL.REL, EXMPL1.REL and creates EXAM.COM.

4.3 FORMAT OF LINK COMPATIBLE OBJECT FILES

NOTE

Section 4.3 is reference material for users who wish to know the load format of LINK-80 relocatable object files. Most users will want to skip this section, as it does not contain material necessary to the operation of the package.

LINK-compatible object files consist of a bit stream. Individual fields within the bit stream are not aligned on byte boundaries, except as noted below. Use of a bit stream for relocatable object files keeps the size of object files to a minimum, thereby decreasing the number of disk reads/writes.

There are two basic types of load items: Absolute and Relocatable. The first bit of an item indicates one of these two types. If the first bit is a 0, the following 8 bits are loaded as an absolute byte. If the first bit is a 1, the next 2 bits are used to indicate one of four types of relocatable items:

00 Special LINK item (see below).

01 Program Relative. Load the following 16 bits after adding the current Program base.

10 Data Relative. Load the following 16 bits after adding the current Data base.

11 Common Relative. Load the following 16 bits after adding the current Common base.

Special LINK items consist of the bit stream 100 followed by:

a four-bit control field

an optional A field consisting of a two-bit address type that is the same as the two-bit field above except 00 specifies absolute address

an optional B field consisting of 3 bits that give a symbol length and up to 8 bits for each character of the symbol

A general representation of a special LINK item is:

```
1 00 xxxx  yy  nn    zzz + characters of symbol name
           ------    ------------------------------
           A field              B field
```

xxxx	Four-bit control field (0-15 below)
yy	Two-bit address type field
nn	Sixteen-bit value
zzz	Three-bit symbol length field

The following special types have a B-field only:

0	Entry symbol (name for search)
1	Select COMMON block
2	Program name
3	Request library search
4	Extension LINK items (see below)

The following special LINK items have both an A field and a B field:

5	Define COMMON size
6	Chain external (A is head of address chain, B is name of external symbol)
7	Define entry point (A is address, B is name)

The following special LINK items have an A field only:

8	External - offset. Used for JMP and CALL to externals
9	External + offset. The A value will be added to the two bytes starting at the current location counter immediately before execution.
10	Define size of Data area (A is size)
11	Set loading location counter to A
12	Chain address. A is head of chain, replace all entries in chain with current location counter. The last entry in the chain has an address field of absolute zero.
13	Define program size (A is size)
14	End program (forces to byte boundary)

The following special Link item has neither an A nor a B field:

15 End file

An Extension LINK item follows the general format of a B-field-only special LINK item, but contents of the B-field are not a symbol name. Instead, the symbol area contains one character to identify the type of Extension LINK item, followed by from 1 to 7 characters of additional information.

Thus, every Extension LINK item has the format:

```
1 00 0100 zzz i jjjjjjj
```

where

zzz	may be any three bit integer (with 000 representing 8),
i	is an eight bit Extension LINK item type identifier, and
jjjjjjj	are zzz-1 eight bit characters of information whose significance depends on i

At present, there is only one Extension LINK item:

i	= X'35' COBOL overlay segment sentinel
zzz	= 010 (binary)
j	= COBOL segment number -49 (decimal)

When the overlay segment sentinel is encountered by the linker, the current overlay segment number is set to the value of j+49. If the previously existing segment number was non-zero and a /N switch is in effect, the data area is written to disk in a file whose name is the current program name and whose extension is Vnn, where nn are the two hexadecimal digits representing the number j+49 (decimal).

4.4 LINK-80 ERROR MESSAGES

LINK-80 has the following error messages:

?No Start Address	A /G switch was issued, but no main program had been loaded.
?Loading Error	The last file given for input was not a properly formatted LINK-80 object file.
?Out of Memory	Not enough memory to load program.
?Command Error	Unrecognizable LINK-80 command.
?<file> Not Found	<file>, as given in the command string, did not exist.

%2nd COMMON Larger /XXXXXX/

The first definition of COMMON block /XXXXXX/ was not the largest definition. Reorder module loading sequence or change COMMON block definitions.

%Mult. Def. Global YYYYYY

More than one definition for the global (internal) symbol YYYYYY was encountered during the loading process.

%Overlaying $\left\{\begin{array}{l}\text{Program}\\ \text{Data}\end{array}\right\}$ Area ,Start = xxxx
,Public = <symbol name>(xxxx)
,External = <symbol name>(xxxx)

A /D or /P will cause already loaded data to be destroyed.

?Intersecting $\left\{\begin{array}{l}\text{Program}\\ \text{Data}\end{array}\right\}$ Area

The program and data area intersect and an address or external chain entry is in this intersection. The final value cannot be converted to a current value since it is in the area intersection.

?Start Symbol - <name> - Undefined

After a /E: or /G: is given, the symbol specified was not defined.

Origin {Above / Below} Loader Memory, Move Anyway (Y or N)?

After a /E or /G was given, either the data or program area has an origin or top which lies outside loader memory (i.e., loader origin to top of memory). If a Y <cr> is given, LINK-80 will move the area and continue. If anything else is given, LINK-80 will exit. In either case, if a /N was given, the image will already have been saved.

?Can't Save Object File

A disk error occurred when the file was being saved.

4.5 PROGRAM BREAK INFORMATION

LINK-80 stores the address of the first free location in a global symbol called $MEMRY if that symbol has been defined by a program loaded. $MEMRY is set to the top of the data area +1.

NOTE

If /D is given and the data origin is less than the program area, the user must be sure there is enough room to keep the program from being destroyed. This is particularly true with the disk driver for FORTRAN-80 which uses $MEMRY to allocate disk buffers and FCB's.

APPENDIX A

TEKDOS Operating System

The command formats for MACRO-80, LINK-80 and CREF-80 differ slightly under the TEKDOS operating system.

A.1 TEKDOS COMMAND FILES

The files F80, M80, and C80 are actually TEKDOS command files for the compiler, assembler, loader, and cross reference programs, respectively. These command files set the emulation mode to 0 and select the Z-80 assembler processor (see TEKDOS documentation), then execute the appropriate program file. You will note that all of these command files are set up to execute the Microsoft programs from drive 1. LINK-80 will also look for the library (FORLIB) on drive 1. If you wish to execute any of this software from drive 0, the command file must be edited and LINK-80 should be given an explicit library search directive "FORLIB-S". (See Section 4.2.1 of this manual.)

A.2 MACRO-80

The M80 assembler accepts command lines only. A prompt is not displayed and interactive commands are not accepted. Commands have the same format as TEKDOS assembler commands; i.e., three filename or device name parameters plus optional switches.

```
M80 [objfile] [lstfile] sourcefile [swl] [sw2...]
```

The object and listing file parameters are optional. These files will not be created if the parameters are omitted, however any error messages will still be displayed on the console. The available switches are as described in Chapter 2 of this manual. except that the switches are delimited by commas or spaces instead of slashes.

A.3 CREF-80

The form of commands to CREF80 is:

```
C80 lstfile sourcefile
```

Both filename parameters are required. The sourcefile parameter is always the name of a CREF80 file created during assembly, by use of the C switch.

Example:

Create a CREF80 file using MACRO-80:

```
M80 ,, TSTCRF TSTMAC C
```

Create a cross reference listing from the CREF80 file:

```
C80 TSTLST TSTCRF
```

A.4 LINK-80

With TEKDOS, the LINK-80 loader accepts interactive commands only. Command lines are not supported.

When LINK-80 is invoked, and whenever it is waiting for input, it will prompt with an asterisk. Commands are lists of filenames and/or devices separated by commas or spaces and optionally interspersed with switches. The input to LINK-80 must be Microsoft relocatable object code (not the same as TEKDOS loader format).

Switches to LINK-80 are delimited by hyphens under TEKDOS, instead of slashes. All LINK-80 switches (as documented in Chapter 3) are supported, except "G" and "N", which are not implemented at this time.

Examples:

1. Assemble a MACRO-80 program named XTEST, creating an object file called XREL and a listing file called XLST:

```
>M80 XREL XLST XTEST
```

2. Load XTEST and save the loaded module:

```
>L80
*XREL-E
[04AD  22B8]
*DOS*ERROR 46
L80 TERMINATED
>M XMOD 400 22B8 04AD
```

Note that "-E" exits via an error message due to execution of a Halt instruction. The memory image is intact, however, and the "Module" command may be used to save it. Once a program is saved in module format, it may then be executed directly without going through LINK-80 again.

The bracketed numbers printed by LINK-80 before exiting are the entry point address and the highest address loaded, respectively. The loader default is to begin loading at 400H. However, the loader also places a jump to the start address in location 0, thereby allowing execution to begin at 0. The memory locations between 0003 and 0400H are reserved for SRB's and I/O buffers at runtime.

INDEX

Microsoft
Software Problem Report

Use this form to report errors or problems in:

- [ ] FORTRAN-80
- [ ] COBOL-80
- [ ] MACRO-80
- [ ] LINK-80

Release (or version) number: __________

Date ______________________

Report only one problem per form.

Describe your hardware and operating system: ___________________

____

Please supply a concise description of the problem and the circumstances surrounding its occurrence. If possible, reduce the problem to a simple test case. Otherwise, include all programs and data in <u>machine readable form</u> (preferably on a diskette). If a patch or interim solution is being used, please describe it.

This form may also be used to describe suggested enhancements to Microsoft software.

Problem Description:

-over-

Did you find errors in the documentation supplied with the software? If so, please include page numbers and describe:

Fill in the following information before returning this form:

Name__________________________________ Phone_____________________

Organization____

Address_______________________ City___________ State____ Zip_____

Return form to: Microsoft
10800 NE Eighth, Suite 819
Bellevue, WA 98004

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

Disclaimer

Table of Contents

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

ED { <filename> | <filename>.<filetype> }

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

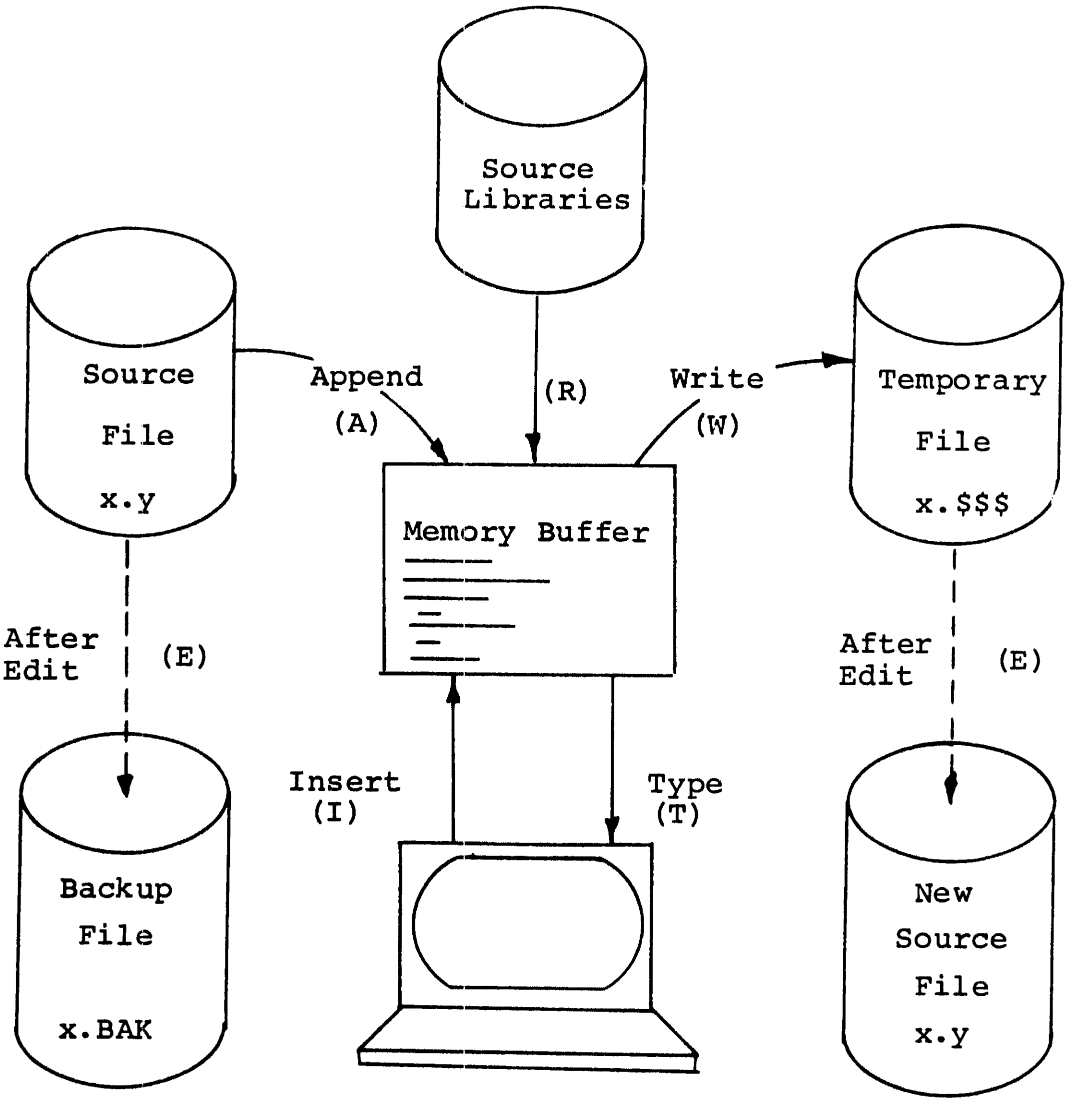
ED operates upon the source file, denoted in Figure 1 by x.y, and passes all text through a memory buffer where the text can be viewed or altered (the number of lines which can be maintained in the memory buffer varies with the line length, but has a total capacity of about 6000 characters in a 16K CP/M 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 alphabetics 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

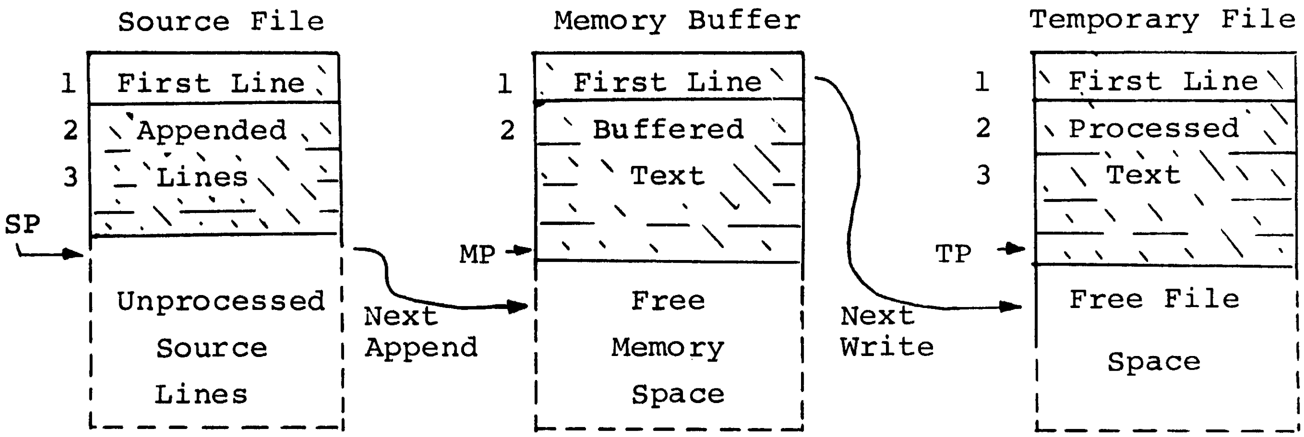
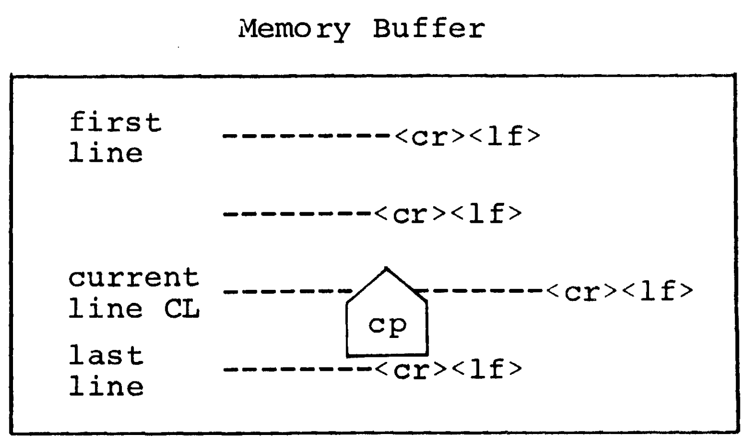


Figure 3. Logical Organization of Memory Buffer



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 id 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 <u>no</u> 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 exceded. 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 cp 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>
cp

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

Command String	Effect	Resulting Memory Buffer
1. B2T<cr>	move to beginning of buffer and type 2 lines: "NOW IS THE TIME FOR"	cp NOW IS THE<cr><lf> TIME FOR<cr><lf> ALL GOOD MEN<cr><lf>
2. 5C0T<cr>	move CP 5 characters and type the beginning of the line "NOW I"	NOW I cp 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>
4.	-L#K<cr>	move up one line, delte 65535 lines which follow	NOW IS THE<cr><lf>
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>
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>
7.	<cr>	move down one line and type one line "INSERT"	NOW IS THE<cr><lf> TIME TO<cr><lf> INSERT<cr><lf>

1.7. Text Search and Alteration

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

$$nF\ c_1 c_2 \ldots c_k \left\{ \begin{matrix} \text{<cr>} \\ \uparrow z \end{matrix} \right\}$$

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:

Command String	Effect	Resulting Memory Buffer
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 $c_1c_2\ldots\ c_n$↑z   or

I $c_1c_2\ldots\ c_n$<cr>

where c_1 through c_n are characters to insert. If the insertion string is terminated by a ↑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:

Command String	Effect	Resulting Memory Buffer
BITHIS IS ↑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 cp 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 cp <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> cp 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

$$n\ S\ c_1c_2\ldots c_k{\uparrow}z\ d_1d_2\ldots d_m \left\{\begin{matrix}\text{<cr>}\\ {\uparrow}z\end{matrix}\right\}$$

and has exactly the same effect as applying the command string

$$F\ c_1c_2\ldots c_k{\uparrow}z\text{-}k\text{DI}d_1d_2\ldots d_m \left\{\begin{matrix}\text{<cr>}\\ {\uparrow}z\end{matrix}\right\}$$

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

$$n\ N\ c_1c_2\ldots c_k \left\{\begin{matrix}\text{cr}\\ {\uparrow}z\end{matrix}\right\}$$

which searches the entire source file for the nth occurrence of the string $c_1c_2\ldots c_k$ (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

$$n\ J\ c_1c_2\ldots c_k\uparrow z\ d_1d_2\ldots d_m\uparrow z\ e_1e_2\ldots e_q \left\{\begin{matrix}\text{<cr>}\\ \uparrow z\end{matrix}\right\}$$

with the following action applied n times to the memory buffer: search from the current CP for the next occurrence of the string $c_1c_2\ldots c_k$. If found, insert the string $d_1,d_2\ldots,d_m$, and move CP to follow d_m. Then delete all characters following CP up to (but not including) the string $e_1,e_2,\ldots e_q$, leaving CP directly after d_m. If $e_1,e_2,\ldots e_q$ cannot be found, then no deletion is made. If the current line is

NOW IS THE TIME<cr><lf>

Then the command

JW ↑zWHAT↑z↑l<cr>

Results in

NOW WHAT <cr><lf>

(Recall that ↑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

$R\ f_1f_2..f_n\uparrow z$ or

$R\ f_1f_2..f_n$<cr>

where $f_1f_2..f_n$ 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<cr>

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:

$$n\ M\ c_1c_2...c_k \begin{Bmatrix} \text{<cr>} \\ \uparrow z \end{Bmatrix}$$

where $c_1c_2...c_k$ 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↑z-5DIDELTA↑z0TT<cr>

or equivalently

MSGAMMA↑zDELTA↑z0TT<cr>

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:

Control Character	Function
↑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)

Command	Function
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)

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

:4ØØT

is interpreted as "type from the current line number through the line whose absolute number is 4ØØ." Combining the two line reference forms, the command

345::4ØØT

for example, is interpreted as "move to absolute line 345, then type through absolute line 4ØØ." Note that absolute line references of this sort can precede any of the standard ED commands.

A special case of the V command, "ØV", 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 ctl-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 disasterous 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.

DIGITAL RESEARCH®

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

CP/M DYNAMIC DEBUGGING TOOL (DDT)

USER'S GUIDE

Disclaimer

Table of Contents

Section | Page

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 loaction 1ØØH 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 1ØØH. 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 8Ø8Ø Assembly Language Reference Card for a list of mnemonics), followed by register references and operands in absolute hexadecimal form. Each sucessive 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 overlayed 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 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 alphabetics 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 comand, with up to two optional breakpoint addresses. The G command takes one ot 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 ØØØH through ØFFH (i.e., the first page of memory). If b is omitted, then b=ØØØØ 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 1ØØH 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

tab character — rubout — rubout echo — Rubout deletes character

```
ED SCAN.ASM
*I
        ORG     100H      L-L;START OF TRANSIENT AREA
        MVI     B,LEN     ;LENGTH OF VECTOR TO SCAN
        MVI     C,0       ;LARGER-RST VALUE SO FAR
LOOP--P-O-O-L   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
;
;       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
*B0P
        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 0608                 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
A>
```

Code/data listing truncated

Value of Equate

```
DDT SCAN.HEX
```
Start Debugger using hex format machine code

```
16K DDT VER 1.0
NEXT  PC
0121 0000
-X
```
last load address +1

next instruction to execute at PC=0

```
C0Z0M0E0I0 A=00 B=0000 D=0000 H=0000 S=0100 P=0000 OUT  7F
-XP
```
Examine registers before debug run

```
P=0000 100
```
Change PC to 100

```
-X
```
Look at registers again

PC changed.

```
C0Z0M0E0I0 A=00 B=0000 D=0000 H=0000 S=0100 P=0100 MVI  B,08
-L100
```
Next instruction to execute at PC=100

```
0100  MVI  B,08
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

C0Z0M0E0I0  A=00  B=0000  D=0000  H=0000  S=0100  P=0100  MVI   B,08
-I    Execute Program for one step.      initial CPU state, before is executed
C0Z0M0E0I0  A=00  B=0000  D=0000  H=0000  S=0100  P=0100  MVI   B,08*0102
-I    Trace one step again (note 08H in B)        automatic breakpoint
C0Z0M0E0I0  A=00  B=0800  D=0000  H=0000  S=0100  P=0102  MVI   C,00*0104
-I    Trace again (Register C is cleared)
C0Z0M0E0I0  A=00  B=0800  D=0000  H=0000  S=0100  P=0104  LXI   H,0119*0107
-T3   Trace three steps
C0Z0M0E0I0  A=00  B=0800  D=0000  H=0119  S=0100  P=0107  MOV   A,M
C0Z0M0E0I0  A=02  B=0800  D=0000  H=0119  S=0100  P=0108  SUB   C
C0Z0M0E0I1  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
0119 02 00 04 03 05 06 01  Program data
0120 05 11 00 22 21 00 02 7E EB 77 13 23 EB 0B 78 B1  ..."!..~.W.#..x.   Lowercase x
0130 C2 27 01 C3 03 29 00 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 00  ................
0150 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  Data is displayed
0160 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  in ASCII with a "."
0170 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  in the position of
0180 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  non-graphic
0190 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  characters
01A0 00 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 00  ................
01C0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
-X          Current CPU state
C0Z0M0E0I1  A=02  B=0800  D=0000  H=0119  S=0100  P=010D  INX   H
-T5   Trace 5 steps from current CPU state
C0Z0M0E0I1  A=02  B=0800  D=0000  H=0119  S=0100  P=010D  INX   H
C0Z0M0E0I1  A=02  B=0800  D=0000  H=011A  S=0100  P=010E  DCR   B      Automatic
C0Z0M0E0I1  A=02  B=0700  D=0000  H=011A  S=0100  P=010F  JNZ   0107   Breakpoint
C0Z0M0E0I1  A=02  B=0700  D=0000  H=011A  S=0100  P=0107  MOV   A,M
C0Z0M0E0I1  A=00  B=0700  D=0000  H=011A  S=0100  P=0108  SUB   C*0109
-U5   Trace without listing intermediate states
C0Z1M0E1I1  A=00  B=0700  D=0000  H=011A  S=0100  P=0109  JNC   010D*0108
-X    CPU State at end of U5
C0Z0M0E1I1  A=04  B=0600  D=0000  H=011B  S=0100  P=0108  SUB   C
```

```
-G↲      Run Program from current PC until completion (in real-time)
*0116    breakpoint at 116H, caused by executing RST 7 in machine code
-X↲      CPU state at end of Program
C0Z1M0E1I1 A=00 B=0000 D=0000 H=0121 S=0100 P=0116 RST  07
-XP↲     examine and change Program counter
P=0116 100↲

-X↲
C0Z1M0E1I1 A=00 B=0000 D=0000 H=0121 S=0100 P=0100 MVI  B,08
-T10↲    Trace 10 (hexadecimal) steps    first data element    current largest value    subtract for comparison A<C
C0Z1M0E1I1 A=00 B=0000 D=0000 H=0121 S=0100 P=0100 MVI  B,08
C0Z1M0E1I1 A=00 B=0800 D=0000 H=0121 S=0100 P=0102 MVI  C,00
C0Z1M0E1I1 A=00 B=0800 D=0000 H=0121 S=0100 P=0104 LXI  H,0119
C0Z1M0E1I1 A=00 B=0800 D=0000 H=0119 S=0100 P=0107 MOV  A,M
C0Z1M0E1I1 A=02 B=0800 D=0000 H=0119 S=0100 P=0108 SUB  C
C0Z0M0E0I1 A=02 B=0800 D=0000 H=0119 S=0100 P=0109 JNC  010D
C0Z0M0E0I1 A=02 B=0800 D=0000 H=0119 S=0100 P=010D INX  H
C0Z0M0E0I1 A=02 B=0800 D=0000 H=011A S=0100 P=010E DCR  B
C0Z0M0E0I1 A=02 B=0700 D=0000 H=011A S=0100 P=010F JNZ  0107
C0Z0M0E0I1 A=02 B=0700 D=0000 H=011A S=0100 P=0107 MOV  A,M
C0Z0M0E0I1 A=00 B=0700 D=0000 H=011A S=0100 P=0108 SUB  C
C0Z1M0E1I1 A=00 B=0700 D=0000 H=011A S=0100 P=0109 JNC  010D
C0Z1M0E1I1 A=00 B=0700 D=0000 H=011A S=0100 P=010D INX  H
C0Z1M0E1I1 A=00 B=0700 D=0000 H=011B S=0100 P=010E DCR  B
C0Z0M0E1I1 A=00 B=0600 D=0000 H=011B S=0100 P=010F JNZ  0107
C0Z0M0E1I1 A=00 B=0600 D=0000 H=011B S=0100 P=0107 MOV  A,M*0108
-A109↲      Insert a "hot patch" into     Program should have moved the
0109 JC 10D↲  the machine code            value from A into C since A>C.
010C↲         to change the               Since this code was not executed,
              JNC to JC                   it appears that the JNC should
-G0↲     Stop DDT so that a version of    have been a JC instruction
         the patched Program can be saved
SAVE 1 SCAN.COM↲     Program resides on first Page, so save 1 page.
A>DDT SCAN.COM↲      Restart DDT with the saved memory image to continue testing
16K DDT VER 1.0
NEXT  PC
0200 0100
-L100↲   List some code

0100  MVI  B,08
0102  MVI  C,00
0104  LXI  H,0119      Previous Patch is Present in X.COM
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
-XP

P=0100

-T10   Trace to see how patched version operates        Data is moved from A to C
C0Z0M0E0I0 A=00 B=0000 D=0000 H=0000 S=0100 P=0100 MVI  B,08
C0Z0M0E0I0 A=00 B=0800 D=0000 H=0000 S=0100 P=0102 MVI  C,00
C0Z0M0E0I0 A=00 B=0800 D=0000 H=0000 S=0100 P=0104 LXI  H,0119
C0Z0M0E0I0 A=00 B=0800 D=0000 H=0119 S=0100 P=0107 MOV  A,M
C0Z0M0E0I0 A=02 B=0800 D=0000 H=0119 S=0100 P=0108 SUB  C
C0Z0M0E0I1 A=02 B=0800 D=0000 H=0119 S=0100 P=0109 JC   010D
C0Z0M0E0I1 A=02 B=0800 D=0000 H=0119 S=0100 P=010C MOV  C,A
C0Z0M0E0I1 A=02 B=0802 D=0000 H=0119 S=0100 P=010D INX  H
C0Z0M0E0I1 A=02 B=0802 D=0000 H=011A S=0100 P=010E DCR  B
C0Z0M0E0I1 A=02 B=0702 D=0000 H=011A S=0100 P=010F JNZ  0107
C0Z0M0E0I1 A=02 B=0702 D=0000 H=011A S=0100 P=0107 MOV  A,M
C0Z0M0E0I1 A=00 B=0702 D=0000 H=011A S=0100 P=0108 SUB  C
C1Z0M1E0I0 A=FE B=0702 D=0000 H=011A S=0100 P=0109 JC   010D
C1Z0M1E0I0 A=FE B=0702 D=0000 H=011A S=0100 P=010D INX  H
C1Z0M1E0I0 A=FE B=0702 D=0000 H=011B S=0100 P=010E DCR  B
C1Z0M0E1I1 A=FE B=0602 D=0000 H=011B S=0100 P=010F JNZ  0107*0107
-X                                                   breakpoint after 16 steps

C1Z0M0E1I1 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

C1Z0M0E1I1 A=04 B=0602 D=0000 H=011B S=0100 P=0108 SUB  C
-T                Single Step for a few cycles

C1Z0M0E1I1 A=04 B=0602 D=0000 H=011B S=0100 P=0108 SUB  C*0109
-T

C0Z0M0E0I1 A=02 B=0602 D=0000 H=011B S=0100 P=0109 JC   010D*010C
-X

C0Z0M0E0I1 A=02 B=0602 D=0000 H=011B S=0100 P=010C MOV  C,A
-G    Run to completion

*0116
-X

C0Z1M0E1I1 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 .
-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
-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
-I
C0Z1M0E1I1 A=03 B=0003 D=0000 H=0121 S=0100 P=0100 MVI  B,08*0102
-I
C0Z1M0E1I1 A=03 B=0803 D=0000 H=0121 S=0100 P=0102 MVI  C,00*0104
-I
C0Z1M0E1I1 A=03 B=0800 D=0000 H=0121 S=0100 P=0104 LXI  H,0119*0107
-I
C0Z1M0E1I1 A=03 B=0800 D=0000 H=0119 S=0100 P=0107 MOV  A,M*0108
```

End of the S command

Review the code

Reset the PC

Single step, and watch data values

Count set

"largest" set

base address of data set

```
-T
                    first data item brought to A
C0Z1M0E1I1 A=02 B=0800 D=0000 H=0119 S=0100 P=0108 SUB  C*0109
-T

C0Z0M0E0I1 A=02 B=0800 D=0000 H=0119 S=0100 P=0109 JC   010D*010C
-T

C0Z0M0E0I1 A=02 B=0800 D=0000 H=0119 S=0100 P=010C MOV  C,A*010D
-T                            first data item moved to C correctly

C0Z0M0E0I1 A=02 B=0802 D=0000 H=0119 S=0100 P=010D INX  H*010E
-T

C0Z0M0E0I1 A=02 B=0802 D=0000 H=011A S=0100 P=010E DCR  B*010F
-T

C0Z0M0E0I1 A=02 B=0702 D=0000 H=011A S=0100 P=010F JNZ  0107*0107
-T

C0Z0M0E0I1 A=02 B=0702 D=0000 H=011A S=0100 P=0107 MOV  A,M*0108
-T                  second data item brought to A

C0Z0M0E0I1 A=00 B=0702 D=0000 H=011A S=0100 P=0108 SUB  C*0109
-T                  subtract destroys data value which was loaded!!!

C1Z0M1E0I0 A=FE B=0702 D=0000 H=011A S=0100 P=0109 JC   010D*010D
-T

C1Z0M1E0I0 A=FE B=0702 D=0000 H=011A S=0100 P=010D INX  H*010E
-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
0109  JC   010D       would not be destroyed.
010C  MOV  C,A
019D  INX  H
010E  DCR  B
010F  JNZ  0107
0112  MOV  A,C
-A108

0108  CMP C        hot patch at 108H changes SUB to CMP

0109

-G0     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                Look at code to see if it was properly Loaded
0119  STAX B             (long typeout aborted with rubout)
011A  NOP
- (rubout)

-G,116     Run from 100H to completion

*0116
-XC        Look at Carry (accidental typo)
C1

-X         Look at CPU state
C1Z1M0E1I1 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
                ctl-Z
           SUB     C          ;LARGER VALUE IN C?
*SSUB↑ZCMP↑Z0LT
           CMP     C          ;LARGER VALUE IN C?
*
           JNC     NFOUND     ;JUMP IF LARGER VALUE NOT FOUND
*SNC↑ZC↑Z0LT
           JC      NFOUND     ;JUMP IF LARGER VALUE NOT FOUND
*E
```

```
ASM SCAN.AAZ
```
Re-assemble, selecting source from disk A
hex to disk A
Print to Z (selects no print file)

```
CP/M ASSEMBLER - VER 1.0

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
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 0B 78 B1 .."!..^.W.#..X.
0130 C2 27 01 C3 03 29 00 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 00 ................
```

\- (rubout) aborts long typeout

```
-G0
```
stop DDT, debug session complete

check to ensure end is still at 116H

SYSTEM CONFIGURATION

The CP/M 2.2 Operating System software is set up for a Xerox 820-II Personal Computer and a Xerox 620 Printer (20 CPS) or 630 Printer (40 CPS). If you have a different configuration, you must go through this procedure to change the CP/M software to work with your system. You can also use the CONFIGUR Utility to modify other segments of CP/M and initialization parameters for adaption of other specific requirements.

The program will present menus from which you can make the appropriate selection for your system; in other words, these instructions are **guidelines** for you to follow -- you'll have to read the information (printed within the instructions **and** on the screen) to make the proper choices.

This procedure contains instructions for the following functions:

- Record Restart Commands
- Select Printer Port Options
- Select Communication Port Options
- Select I/O Device Assignments
- Select Keyboard Data Format
- Select Screen Attributes
- Select Floppy Disk Head Step Rate*

  or

  Configure Rigid Disk*

* The Floppy Head Step Rate is not recognized or used by systems configured with a rigid disk drive. The floppy head step rate is only recognized and used by systems with floppy disk drives only.

To use the CONFIGUR Utility, you must first load the CP/M software and CONFIGUR Utility, then select the function you wish to use.

When using the CONFIGUR Utility with single density disks, you can only temporarily save any configuration changes.

To configure the rigid disk, you must format (initialize) it first. (Format will erase all the data on the rigid disk.)

To load the CP/M software and CONFIGUR Utility, use the following instructions:

TURN ON the 820-II, or press the **RESET** button or CTRL + ESC if it is already on. (The locations of the ON/OFF switch and RESET button are shown in your CP/M Handbook, along with instructions on how to insert a disk.)

Result The screen will display the following:

```
820-II v 0.00 (C) 1982 Xerox Corp

L - Load System
H - Host Terminal
T - Typewriter

*
```

INSERT the CP/M System Disk in the left (A) disk drive.

TYPE the letters **LA** and press the RETURN key.

Result After a few seconds, the screen displays:

```
60k CP/M vers 2.20 #2-294 DCT0000000
A
```

TYPE the word **CONFIGUR** and press the RETURN key.

Result The following message will be displayed on the screen:

```
          System Configuration -- Version 0.00
          Copyright (c) 1982, XEROX Corporation


Enter SOURCE disk name  (or  RETURN  for this disk):
```

TYPE the name of the disk drive which already has the parameters you wish to use.

OR

press the RETURN key to use or change the parameters of the currently selected disk drive.

Result The following message is displayed on the screen:

```
Enter DESTINATION disk name  (or  RETURN  for same disk)
```

TYPE the disk drive name of the disk containing the operating system that you want modified.

OR

press the RETURN key to modify the operating system on the disk of the currently selected disk drive.

Result The main menu will be displayed as follows:

```
              System Configuration -- Version 0.00
             Copyright (c) 1982, XEROX Corporation

Enter number of function to be executed. Upon completion of the
selected function the program will return to this menu.
Touch  ESC  when all desired selections have been made.

1)    Record Restart Command
2)    Select Printer Port Options
3)    Select Comm Port Options
4)    Select I/O Device Assignments
5)    Select Keyboard Data Format
6)    Select Screen Attributes
7)    Select Floppy Disk Head Step Rate (floppy disk only)
      or
      Configure Rigid Disk (rigid disk only)
```

To use the CONFIGUR Utility function(s), choose the function, then go to the page as indicated below and follow the instructions.

Page 5

Record Restart Command. This allows you to enter a CP/M command (such as DIR, STAT, etc.) to be executed automatically each time you load CP/M.

Page 6

Select Printer Port Options. This allows you to change the standard printer port options (such as Baud Rate, Stop Bits, etc.).

Page 8

Select Communication Port Options. This allows you to change the standard communications port options (such as Baud Rate, Stop Bits, etc.).

Page 10

Select I/O Device Assignments. This allows you to select I/O device assignments (such as Console, etc.).

Page 12

Select Keyboard Data Format. This allows you to change the keyboard data format (7 or 8 bits).

Page 14

Select Screen Attributes. This allows you to select blink, inverse video, high-light, etc., on your screen.

Page 16

Select Floppy Disk Head Step Rate. This allows you to select a step rate (speed) at which **your** disk drives will operate.

Page 18

Configure Rigid Disk. This allows you to divide the rigid disk into four disks. You have the option of selecting one of four recommended configurations or you may choose to divide the rigid disk space in a way that will meet your own specific requirements.

Note: Before you can configure the rigid disk, you must first put the operating system in one of the rigid disk partitions. If the operating system is not on the rigid disk, use your CP/M Handbook and SYSGEN Utility to copy the operating system.

Record Restart Command

TYPE the number **1** to select the "Record Restart Command" function.

Result The following message will be displayed on the screen:

```
        Note:  The selection on this page is saved permanently
               and immediately upon entry - Be careful !
----------------------------------------------------------------------
Enter the normal CP/M command you wish to be executed after every
"COLD" boot (Power up,  CTRL+ESC,  or restart button)

                   End your entry with a  RETURN.

                               -OR-

     Enter either a  RETURN  or an ESC  to clear the command.
```

TYPE the CP/M command that you want and press the RETURN key.

OR

press the RETURN or ESC key to clear the command.

Result The main menu will be displayed on the screen:

IF you need to use another function, turn to page 4 in this section and follow the instructions.

PRESS the ESC key when you have completed all of your selections.

Result The following message is displayed on the screen:

```
The selections you have made are not yet in effect.

Please enter one of the following:

          S  =  Save and activate the selections permanently.

Q  or  ESC  =  Quit without making any changes.

          T  =  Temporarily activate the selections - but do not save them.
```

CHOOSE one of the above selections to return to the operating system.

Select Printer Port Options

TYPE the number **2** to select the "Select Printer Port Options" function.

Result The current printer parameters will be displayed as follows:

Move cursor to option and SPACE to cycle through valid parameters.

Touch ESC to exit or R to reset to original values.

OPTION	PRINTER
Baud Rate	1200
Stop Bits	1
Word Length	7
Parity	Even
Clear-to-Send	Ignore
Carrier Detect	Ignore
Protocol	XON/XOFF

PRESS the SPACE BAR to step through the parameters. When the correct parameter is displayed, go to the next step.

PRESS the DOWN ARROW key once to move to the next option and use the SPACE BAR to step through the parameters. Continue this procedure until you have completed selecting all of the options required to initialize your printer port before going to the next step.

CHECK your screen to be sure that you have selected all of the options needed for your printer.

PRESS the ESC key to display the main menu when you have finished making all of your selections.

OR

type **R** to reset the parameters to their original values and then press the ESC key to display the main menu.

IF you need to use another function, turn to page 4 in this section and follow the instructions.

PRESS the ESC key when you have completed all of your selections.

Result The following message is displayed on the screen:

```
The selections you have made are not yet in effect.

Please enter one of the following:

         S = Save and activate the selections permanently.

Q or ESC = Quit without making any changes.

         T = Temporarily activate the selections - but do not save them.
```

CHOOSE one of the above selections to return to the operating system.

The printer port options shown, work with the Xerox 620 Printer (20 CPS) and 630 Printer (40 CPS).

To select the correct options for other printers, see the printer manual that came with your printer.

Select Communication Port Options

TYPE the number **3** to select the "Select Comm Port Options" function.

Result The current Comm parameters will be displayed as follows:

```
Move cursor to option and  SPACE  to cycle through valid parameters.
Touch  ESC  to exit or  R  to reset to original values.

OPTION                    COMM

 Baud Rate                  300

 Stop Bits                    1

 Word Length                  7

 Parity                    None
```

PRESS the SPACE BAR to step through the parameters. When the correct parameter is displayed, go to the next step.

PRESS the DOWN ARROW key to move to the next option and use the SPACE BAR to step through the parameters. Continue this procedure until you have completed selecting all the options required to initialize your comm port before going to the next step.

CHECK your screen to be sure that you have selected all of the options needed for your communications port.

PRESS the ESC key to display the main menu when you have finished making all of your selections.

OR

type the letter **R** to reset the parameters to their original values and then touch the ESC key to display the main menu.

IF you need to use another function, turn to page 4 in this section and follow the instructions.

PRESS the ESC key when you have completed all of your selections.

Result The following message is displayed on the screen:

```
The selections you have made are not yet in effect.


Please enter one of the following:

         S  =  Save and activate the selections permanently.

Q  or  ESC  =  Quit without making any changes.

         T  =  Temporarily activate the selections - but do not save them.
```

CHOOSE one of the above selections to return to the operating system.

Select I/O Device Assignments

TYPE the number 4 to select the "Select I/O Device Assignments" function.

Result The current device assignments will be displayed on the screen as follows:

```
Move the cursor to device and SPACE through allowed assignments.

CRT = Display and Keyboard, LPT = Printer, COMM = Communications

Touch ESC to exit or R to reset to original values.

Output Device          CP/M and XEROX Designations

Console    =           CRT:

List       =           LPT:      (Serial Printer)
```

Note: To use the parallel printer (list device) option, the parallel printer cable and jumpers must be installed on the 820-II processor (see Parallel Printer Installation Kit in the System Components section of this manual).

PRESS the SPACE BAR to step through the parameters. When the correct parameter is displayed, go to the next step.

PRESS the DOWN ARROW key to move to the next option and press the SPACE BAR to step through the parameters. Continue this procedure until you have completed selecting.

CHECK your screen to be sure that you have selected all of the device assignments needed for your system.

PRESS the ESC key to display the main menu when you have finished making all of your selections.

OR

type the letter **R** to reset the parameters to their original values and then press the ESC key to display the main menu.

IF you need to use another function, turn to the appropriate page in this section and follow the instructions.

PRESS the ESC key when you have completed all of your selections.

Result The following message is displayed on the screen:

```
The selections you have made are not yet in effect.

Please enter one of the following:

          S  =  Save and activate the selections permanently.

Q  or  ESC  = Quit without making any changes.

          T  =  Temporarily activate the selections - but do not save them.
```

CHOOSE one of the above selections to return to the operating system.

Select Keyboard Data Format

TYPE the number **5** to select the "Select Keyboard Data Format" function.

Result The current keyboard data format (configuration) will be displayed on the screen as follows:

```
SPACE  to select keyboard configuration.

Touch  ESC  to exit or  R  to reset to original value.

Keyboard Data Format    7  Bits
```

PRESS the SPACE BAR to select keyboard configuration.

CHECK your screen to be sure that the correct keyboard configuration is displayed.

PRESS the ESC key to display the main menu when you have finished making your selection.

OR

type the letter **R** to reset the parameter to the original value and then press the ESC key to display the main menu.

IF you need to use another function, turn to the appropriate page in this section and follow the instructions.

PRESS the ESC key when you have completed all of your selections.

Result The following message is displayed on the screen:

The selections you have made are not yet in effect.

Please enter one of the following:

S = Save and activate the selections permanently.

Q or ESC = Quit without making any changes.

T = Temporarily activate the selections - but do not save them.

CHOOSE one of the above selections to return to the operating system.

Select Screen Attributes

TYPE the number **6** to select the "Select Screen Attributes" function.

Result The current screen attributes will be displayed on the screen:

```
SPACE  to select desired screen attribute.

Touch  ESC  to exit or  R  to reset to original value.


Screen Attribute    Blink
```

PRESS the SPACE BAR to select a screen attribute.

CHECK your screen to be sure that the correct screen attribute is displayed.

PRESS the ESC key to display the main menu when you have finished making your selection.

OR

type the letter **R** to reset the parameter to the original value and then press the ESC key to display the main menu.

IF you need to use another function, turn to the appropriate page in this section and follow the instructions.

PRESS the ESC key when you have completed all of your selections.

Result The following message is displayed on the screen:

The selections you have made are not yet in effect.

Please enter one of the following:

S = Save and activate the selections permanently.

Q or ESC = Quit without making any changes.

T = Temporarily activate the selections - but do not save them.

CHOOSE one of the above selections to return to the operating system.

Select Floppy Disk Head Step Rate

TYPE the number **7** to select the "Select Floppy Disk Head Step Rate" function.

Result The current floppy disk drive step rate will be displayed on the screen:

```
SPACE  to select floppy disk drive step rate.

Touch  ESC  to exit or  R  to reset to original value.


Drive Step Rate   15  msec.
```

PRESS the SPACE BAR to select floppy disk drive step rate.

Note: The following disk drive head step rates are recommended for use with Xerox disk drives.

DISK DRIVES	SERIAL NUMBER	STEP RATE
8" Double Sided	F10-000-0000	15 msec
8" Single Sided	X973-000-0000	15 msec
5¼" Double Sided	T66-000-0000	30 msec
5¼" Single Sided	X929-000-0000	30 msec

CHECK your screen to be sure that the correct step rate has been selected.

PRESS the ESC key to display the main menu when you have finished making your selection.

OR

type the letter **R** to reset the parameter to the original value and then touch the ESC key to display the main menu.

IF you need to use another function, turn to the appropriate page in this section and follow the instructions.

PRESS the ESC key when you have completed all of your selections.

Result The following message is displayed on the screen:

```
The selections you have made are not yet in effect.

Please enter one of the following:

          S  =  Save and activate the selections permanently.

Q  or  ESC   =  Quit without making any changes.

          T  =  Temporarily activate the selections - but do not save them.
```

CHOOSE one of the above selections to return to the operating system.

Configure Rigid Disk

There are three standard rigid disk partition (space) configurations to choose from or, you may custom partition the rigid disk into as many as four logical disks.

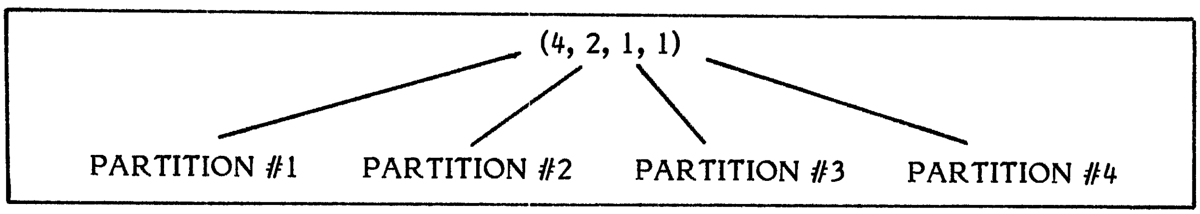
Standard Partitions

Partition #	Size K	Partition #	Size K	Partition #	Size K	Partition #	Size K	
1	= 4032	2	= 1984	3	= 960	4	= 960	*(4,2,1,1)

Partition #	Size K	Partition #	Size K	Partition #	Size K	Partition #	Size K	
1	= 1984	2	= 1984	3	= 1984	4	= 1984	*(2,2,2,2)

Partition #	Size K	Partition #	Size K	Partition #	Size K	Partition #	Size K	
1	= 7936	2	= 0	3	= 0	4	= 0	*(8,0,0,0)

* Rigid disk configuration (size of partitions in megabytes).



Custom Partitioning

When custom partitioning, the smallest amount of space that you are allowed to allocate to a partition (logical drive) is 128K. Any additional space added will be in 64K increments. Listed below are the drive names (logical) for each partition.

LOGICAL NAME	PARTITION NUMBER
E	1
F	2
G	3
H	4

Reconfiguring the Rigid Disk

Use the following table when configuring or reconfiguring the rigid disk:

FROM	TO	CONDITIONS
(4,2,1,1)	(2,2,2,2)	If drives F,G and H are unused (no files) and drive E has less than 2 megabytes of space used.
	(8,0,0,0)	This can only be selected on a newly formatted (blank) disk.
(2,2,2,2)	(4,2,1,1)	Only if F,G and H are unused (no files).
	(8,0,0,0)	This can only be selected on a newly formatted (blank) disk.
(8,0,0,0)	(4,2,1,1)	This can only be selected on a newly formatted disk.
	(2,2,2,2)	This can only be selected on a newly formatted disk.
CUSTOM	(4,2,1,1) (2,2,2,2) (8,0,0,0)	A newly formatted disk must be used to reconfigure from a custom configuration to a standard configuration.

Note: The rigid disk cannot be configured or reconfigured if drive H has been used.

TYPE the number **7** to select the "Configure Rigid Disk" function.

Result The current rigid disk configuration will be displayed on the screen:

```
Use  SPACE  to cycle through the standard rigid disk partition allocations.

Touch  ESC  to exit or  R  to reset to original values.

Partition  Size  Partition  Size  Partition  Size  Partition  Size
    #        K       #        K       #        K       #        K
---------- ----- ---------- ----- ---------- ----- ---------- -----

    1    =  4032     2    =  1984     3    =  960      4    =  960  (4,2,1,1)
```

PRESS the SPACE BAR to select a standard partition allocation or to select the custom partition program.

IF you've selected a standard partition allocation, press the ESC key to exit or type the letter R to reset to the original allocation.

OR

IF you chose to select the custom partition program, press the RETURN key to display the following menu:

```
Move the cursor and  SPACE  to select the storage to be allocated to any
partition.  Once a new partition is allocated, the allocation(s) for
those preceding it may not be changed.  (A new partition is allocated
by dividing the last partition.)  Touch  ESC  to exit or  R  to reset.

DISK ASSIGNMENT               ALLOCATION

  Partition 1:                    7936K

  Partition 2:                    Not Allocated

  Partition 3:                    Not Allocated

  Partition 4:                    Not Allocated
```

PRESS the SPACE BAR to allocate space to partition 2. When you've allocated the amount of space that you want in partition 2, go on to the next step.

PRESS the down arrow key to move to partition 3 and press the SPACE BAR to allocate the amount of space to partition 3. Continue this procedure until you have completed allocating space.

CHECK your screen to be sure that you have completed allocating space.

PRESS the ESC key twice to display the main menu when you have finished making all of your selections.

OR

type the letter **R** to reset the partitions to their original values and then press the ESC key twice to display the main menu.

IF you need to use another function, turn to the appropriate page in this section and follow the instructions.

PRESS the ESC key when you have completed all of your selections.

Result The following message is displayed on the screen:

```
The selections you have made are not yet in effect.

Please enter one of the following:

            S  =  Save and activate the selections permanently.

Q  or  ESC  =  Quit without making any changes.
```

Note: Rigid disk drive selections cannot be temporarily saved.

CHOOSE one of the above selections to return to the operating system.

BACKUP

The BACKUP Utility is a procedure used to backup (copy) the files (data) that are stored on the rigid disk.

The program will present menus from which you can make the appropriate selection; in other words, these instructions are **guidelines** for you to follow -- you'll have to read the information (printed within the instructions **and** on the screen) to make the proper choices.

This utility contains instructions for the following options:

- List Directory
- Backup Files
- Replace Files
- Verify Disk Integrity
- Delete Files
- Exit to CP/M

To use the BACKUP Utility options, you must first load the CP/M software and BACKUP Utility, then select the option you wish to use.

To load the CP/M software and BACKUP Utility, use the following instructions:

TURN ON the 820-II, or press the **RESET** button or CTRL + ESC if it is already on. (The locations of the ON/OFF switch and RESET button are shown in your CP/M Handbook, along with instructions on how to insert a disk.)

Result The screen will display the following:

```
820-II v 0.00 (C) 1982 Xerox Corp

L - Load System
H - Host Terminal
T - Typewriter

*
```

INSERT the CP/M System Disk in the left (A) disk drive.

TYPE the letters **LA** and then press the RETURN key.

Result After a few seconds the screen displays:

```
Xerox 60k CP/M vers 2.20 #2-294 DCT000000
A
```

TYPE the word **BACKUP** and then press the RETURN key.

Result The disk backup and maintenance menu will be displayed on the screen as shown below:

```
                 The Xerox Disk Backup & Maintenance System
                 (C) 1982 Balcones Computer Corporation (P)
                 All Rights Reserved  Version 0.00 (Month) 1982

                      DISK BACKUP & MAINTENANCE MENU

          You have the following options available:

     (1)  List Directory
     (2)  Backup Files
     (3)  Replace Files
     (4)  Verify Disk Integrity
     (5)  Delete Files
     (6)  Exit to CP/M

          Please enter your choice: ( _ )
```

Note: When using the BACKUP Utility, you may want to increase the brightness of your screen using the brightness control located under the left edge of the screen.

To use the BACKUP Utility options, choose the option as shown on the next page, then go to the page as indicated and follow the instructions.

List Directory. The List Directory option allows you to examine the disk directory and to print it out on the printer if you have one.

GO TO PAGE 25

Backup Files. This option allows any or all files to be copied from the rigid disk to floppy disks and store them under a unique backup (session) name.

Note: If any of the files that are being backed up are larger than the space of the disk they are being backed up onto, use the procedure for large files. A single sided double density disk has 480K of available space and a double sided double density disk has 978K.

GO TO PAGE 26

GO TO PAGE 32 (large file procedure)

Replace Files. The Replace Files option operates the opposite of the Backup option. The files (data) are copied from the backup disks (floppies) using the unique backup session name to the rigid disk.

Note: Check to be sure that the disk (rigid partition) you are replacing your files on, is large enough to accept all of the files. A backup session may have required a number of disks.

GO TO PAGE 38

GO TO PAGE 41 (large file procedure)

Verify Disk Integrity. This option allows the verification of the surface of the disk. When flawed sectors are located, the Verify Disk Integrity option will identify those sectors and prevent them from being used by allocating them to a user area.

GO TO PAGE 43

Delete Files. The Delete Files option allows the deletion of files while using the BACKUP Utility without having to return to the operating system.

GO TO PAGE 44

Exit to CP/M. This option allows you to return to the CP/M operating system.

GO TO PAGE 46

List Directory

TYPE the number **1**

Result The following message is displayed on the screen:

```
The Xerox Disk Backup & Maintenance System
(C) 1982 Balcones Computer Corporation (P)
All Rights Reserved Version 0.00 (Month) 1982

LIST DIRECTORY

Which Disk Drive (A-P): ( _ )
```

TYPE the name of the disk drive that you want to examine the directory of.

Result The following message is displayed on the screen:

```
Indicate User Area (00-31): ( _ _ )
```

TYPE the number of the user area (00 to 30).

Result The following message is displayed on the screen:

```
Do you want it listed on the PRINTER also (Y or N): ( _ )
```

TYPE the letter **Y** to print the directory on the printer.

OR

the letter **N** to display the directory on the screen only.

Note: User area 31 is reserved for allocating any flawed (bad) sectors that may be found on the rigid disk when using the Verify Disk Integrity option.

Result The directory for the disk drive and user area that was entered is displayed on the screen.

PRESS any key on the keyboard to return to the Disk Backup and Maintenance Menu.

Backup Files

Note: Check to be sure that you have enough floppy disks initialized (prepared for use) before starting.

TYPE the number **2**

Result The following message is displayed on the screen:

```
The Xerox Disk Backup & Maintenance System
(C) 1982 Balcones Computer Corporation (P)
All Rights Reserved Version 0.00 (Month) 1982

BACKUP FILES

Note: Touch ESC any time to exit.

Enter Disk Drive that Files are to be copied FROM (A-P): ( _ )
```

TYPE the name of the disk drive (A through P) which contains the files you want to backup (copy).

Result The following message is displayed on the screen:

```
Indicate User Area (00-31): ( _ _ )
```

Note: User area 0 (00) is the most commonly used, and on most systems is reserved for system and applications programs. User areas 2 (02) through 30 may be used to create and store files in to assist in file (data) organization. User area 31 is reserved for use by the Verify Disk Integrity option and is never accessed by any program.

TYPE the user area number (0 through 30).

Result The following message is displayed on the screen:

```
Enter Disk Drive that Files are to be copied TO (A-P): ( _ )
```

TYPE the name (A through P) of the floppy disk drive.

Result The following message is displayed on the screen:

```
Indicate User Area (00-31): ( _ _ )
```

TYPE the name of the user area that the files are to be copied in.

Result The following message is displayed on the screen to allow you to check your entries:

```
Copy files FROM drive ( _ ), user ( _ ) TO drive ( _ ), user ( _ ) - Correct (Y or N)?
```

TYPE the letter **Y** to accept.

OR

the letter **N** if you made an error to exit the Backup Files option.

IF you typed the letter **Y**, the following message is displayed on the screen:

```
When Disk is ready in Drive ( _ ), touch any key to continue.
```

WHEN the disk to be copied FROM is ready, press any key on the keyboard to continue.

Result The following message is displayed on the screen:

```
Enter the file name and touch RETURN:
```

TYPE the name of each file to be copied pressing the RETURN key once after each file name is entered.

OR

. to copy all of the files on the disk.

***.COM** to copy only the files that have the file name extension COM.

.*-COM*. to copy all files except the files that have the file extension COM.

Note: Any file name extension may be used to select files.

Result The files to be copied and the following message is displayed on the screen:

Touch RETURN if you are finished entering file names.

PRESS the RETURN key.

Result The following message is displayed on the screen:

Enter backup session name and touch RETURN:

TYPE the name that you want the backup (copy) session to be called, then touch the RETURN key.

Result The system will load the program and after a few seconds the following message is displayed on the screen:

When disk to copy FROM is ready in drive (_) touch RETURN (or ESC to exit)

PRESS the RETURN key.

Result The system will compute the total amount of space required to backup the files that were entered and then display the following message:

When disk to copy TO is ready in drive (_) touch RETURN (or ESC to exit)

INSERT an initialized disk in drive A, then press the RETURN key.

Result The following message will be displayed on the screen:

```
 00K    Remaining to be copied from ( ): User (_ _).
 000K   Available for use  on drive ( ): User (_ _). Better get some more diskettes
r 00    Copying file  (filename) . 000k
```

Note: The message "Better get some more diskettes" will only be displayed on the screen as shown in the above example if the disk you are copying to doesn't have enough space to store all of the files that are being copied.

IF the disk you are copying to has enough space to store all of the files, the following message will be displayed when copying is complete:

```
Touch any key to exit. . .
```

Note: You would remove the backup disk and insert the CP/M system disk. Then press any key on the keyboard to return to te Disk Backup and Maintenance Menu.

IF all of the files cannot fit on the disk in drive A, when the disk cannot fit any more files, the following message is displayed on the screen:

```
 000 K Remaining to be copied from ( ): User (_ _).
There is not enough space on the disk in drive ( ) to copy any more files.

 (1)  Oh no, Drive ( ) is a Non-Removable Hard Disk
 (2)  Try to fit the Remaining Files on another Diskette in Drive ( )
 (3)  Split  (filename)  (000K)  over Multiple Diskettes in Drive ( )
                                            DO NOT REMOVE ANY DISKS YET
      Please enter your choice (or  ESC   to exit)
```

Note:
- The first (1) option will return you to the Disk Backup and Maintenance Menu.
- The second (2) option is the normal choice you would use. Starting with the largest file, the remaining files will be copied on the next disk until all of the files have been copied or the disk is full. This option can only be used when backing up onto floppy disks.
- The third (3) option is used to backup a file that will not fit on one disk.

TYPE the number 2

Result The following message is displayed on the screen:

```
000 K  Remaining to be copied from ( ):  User ( )
When disk to copy TO is ready in drive ( ) touch RETURN  (or ESC to exit)
```

Note: Write the following information on the disk label **before** putting it on the disk. Writing on the label after it has been put on a disk, could damage the disk.

REMOVE and label the disk providing the following information which will be needed when replacing files:

- Disk number (Number the disks as they are copied 1, 2 etc.).)
- Backup session name.
- User area.

INSERT an initialized disk, then press the RETURN.

Result The following message will be displayed on the screen:

```
00K    Remaining to be copied from ( ): User (__).
000K   Available for use  on drive ( ): User (__). Better get some more diskettes

r 00   Copying file  (filename)
```

IF the disk you are copying to has enough space to store the remaining files, the following message will be displayed when copying is complete:

```
Touch any key to exit. . .
```

Note: You would remove the backup disk and insert the CP/M system disk. Then press any key on the keyboard to return to te Disk Backup and Maintenance Menu.

IF the remaining files cannot fit on the disk in drive ( ), when the message that tells you there is not enough space left on the disk displays on the screen, repeat the steps starting at the top of this page until all of the files have been copied. The message "Touch any key to exit . . . " will be displayed on the screen when copying is complete.

PRINT the disk directories and keep them with each disk. The directories will be needed when the files are replaced.

Note: The Backup Files Option requires space on the disk to record (save) the backup session information which is used when replacing the files. This information is saved on the disk(s) and is identified by the file extension name **BKD. DO NOT** delete this file from the disk.

Backup Files (Large file procedure)

Note: Check to be sure that you have enough floppy disks initialized (prepared for use) before starting.

TYPE the number **2**

Result The following message is displayed on the screen:

```
                The Xerox Disk Backup & Maintenance System
                (C) 1982 Balcones Computer Corporation (P)
                All Rights Reserved Version 0.00 (Month) 1982

                              BACKUP FILES

     Note: Touch ESC any time to exit.

     Enter Disk Drive that Files are to be copied FROM (A-P): ( _ )
```

TYPE the name of the disk drive (A through P) which contains the files you want to backup (copy).

Result The following message is displayed on the screen:

```
                    Indicate User Area (00-31): ( _ _ )
```

Note: User area 0 (00) is the most commonly used, and on most systems is reserved for system and applications programs. User areas 2 (02) through 30 may be used to create and store files in to assist in file (data) organization. User area 31 is reserved for use by the Verify Disk Integrity option and is never accessed by any program.

TYPE the user area number (0 through 30).

Result The following message is displayed on the screen:

```
     Enter Disk Drive that Files are to be copied TO (A-P): ( _ )
```

TYPE the name (A through P) of the floppy disk drive.

Result The following message is displayed on the screen:

Indicate User Area (00-31): (_ _)

TYPE the name of the user area that the files are to be copied in.

Result The following message is displayed on the screen to allow you to check your entries:

Copy files FROM drive (_), user (_) TO drive (_), user (_) - Correct (Y or N)?

TYPE the letter **Y** to accept.

OR

the letter **N** if you made an error to exit the Backup Files option.

IF you typed the letter **Y**, the following message is displayed on the screen:

When Disk is ready in Drive (_), touch any key to continue.

WHEN the disk to be copied FROM is ready, press any key on the keyboard to continue.

Result The following message is displayed on the screen:

Enter the file name and touch RETURN:

TYPE the name of the file that is larger than the disk that you are copying to.

Result The file name that was entered and the following message is displayed on the screen:

Touch RETURN if you are finished entering file names.

PRESS the RETURN key.

Result The following message is displayed on the screen:

Enter backup session name and touch RETURN:

TYPE the name that you want the backup (copy) session to be called, then touch the RETURN key.

Result The system will load the program and after a few seconds the following message is displayed on the screen:

When disk to copy FROM is ready in drive (_) touch RETURN (or ESC to exit)

PRESS the RETURN key.

Result The system will compute the total amount of space required to backup the file that was entered and then display the following message:

When disk to copy TO is ready in drive (_) touch RETURN (or ESC to exit)

INSERT an initialized disk in drive A, then press the RETURN key.

Result The following message will be displayed on the screen:

```
 000 K Remaining to be copied from ( ): User (_ _).
There is not enough space on the disk in drive ( ) to copy any more files.

 (1)  Oh no, Drive ( ) is a Non-Removable Hard Disk
 (2)  Try to fit the Remaining Files on another Diskette in Drive ( )
 (3)  Split (filename) (000K) over Multiple Diskettes in Drive ( )
                                        DO NOT REMOVE ANY DISKS YET
      Please enter your choice (or  ESC  to exit)
```

Note:
- The first (1) option will return you to the Disk Backup and Maintenance Menu.
- The second (2) option is the normal choice you would use. Starting with the largest file, the remaining files will be copied on the next disk until all of the files have been copied or the disk is full. This option can only be used when backing up onto floppy disks.
- The third (3) option is used to backup a file that will not fit on one disk.

TYPE the number 3

Result The following message is displayed on the screen:

```
( ) : (filename) .

 000 K Remaining to be copied from ( ): User (_ _)

Note: Splitting a file into pieces requires that drive ( ) has removable
media. Do not change diskettes until prompted to do so.
Be sure and label the diskettes copied as part 1, part 2, etc.

                                        DO NOT REMOVE ANY DISKS YET
Are you sure you want to split file (filename) . (Y or  ESC  to exit):
```

TYPE the letter **Y**

Result The following message is displayed on the screen:

```
( ) : (filename) .

 000 K Remaining to be copied from ( ): User (_ _).
 000 K Available for use on drive ( ): User (_ _). Better get some more diskettes.
Note: Splitting a file into pieces requires that drive ( ) has removable
media. Do not change diskettes until prompted to do so.
Be sure and label the diskettes copied as part 1, part 2, etc.

When a disk is ready in drive ( ) for (filename) . part 1, type  RETURN
```

PRESS the RETURN key.

Result The following message is displayed on the screen:

```
r 000 Copy 000 K out of 000 K (filename) . Part 1
```

WHEN the disk in physical drive A is full, the following message is displayed on the screen:

```
When a disk is ready in drive ( ) for (filename) . part 2, type RETURN
```

Note: Write the following information on the disk label **before** putting it on the disk. Writing on the label after it has been put on a disk, could damage the disk.

REMOVE and label the disk with the following information which will be needed when replacing the file:

- Disk number (Number the disks as they are copied 1, 2, etc.).
- Backup session name.
- User area.

INSERT an initialized disk, then press the RETURN key.

Result The following message will be displayed on the screen:

```
 00 K Remaining to be copied from ( ): User (_ _).
 000 K Available for use on drive ( ): User (_ _).
Note: Splitting a file into pieces requires that drive ( ) has removable
media. Do not change diskettes until prompted to do so.
Be sure and label the diskettes copied as part 1, part 2, etc.

r 00 Copy 000 K out of 000 K (filename) . Part 2
```

IF the disk you are copying to has enough space to store the remaining part of the file being copied, the following message will be displayed when copying is complete:

```
( ) : (filename) .

0 K      Remaining to be copied from ( ): User (_ _).
000 K    Available for use on drive ( ): User (_ _).

When disk to copy TO is ready in drive ( ) touch    RETURN  (or ESC to exit)
```

Note: You would press the RETURN key and the following message would be displayed on the screen:

```
Touch any key to exit. . .
```

Note: You would remove the backup disk and insert the CP/M system disk. Then press any key on the keyboard to return to the Disk Backup And Maintenance Menu.

IF the remaining part of the file cannot fit on the disk, when the message that tells you there is not enough space left on the disk displays on the screen, repeat the steps on the previous page starting with "REMOVE and label the disk etc." until all of the file has been copied. The message "Touch any key to exit . . ." will be displayed on the screen.

PRINT the disk directories and keep them with each disk.

Note: The Backup Files Option requires space on the disk to record (save) the backup session information which is used when replacing the files. This information is saved on the disk(s) and is identified by the file extension name **BKD. DO NOT** delete this file from the disk.

Replace Files

CHECK to be sure that the rigid disk partition that will be used, has enough space available before starting.

TYPE the number **3**

Result The following message is displayed on the screen:

```
The Xerox Disk Backup & Maintenance System
(C) 1982 Balcones Computer Corporation (P)
All Rights Reserved Version 0.00 (Month) 1982

REPLACE FILES

Note: Touch ESC any time to exit.

Enter Disk Drive that Files are to be copied FROM (A-P): ( _ )
```

TYPE the name of the disk drive that the files are to be copied from.

Result The following message is displayed on the screen:

```
Indicate User Area (00-31): ( _ )
```

TYPE the user area number that the files are to be copied from.

Result The following message is displayed on the screen:

```
Enter Disk Drive that Files are to be copied TO (A-P): ( _ )
```

TYPE the name of the disk drive that the files are to be copied to.

Result The following message is displayed on the screen:

```
Indicate User Area (00-31): ( _ )
```

TYPE the user area number that the files are to be copied to.

Result The following message is displayed on the screen:

Copy files FROM drive (_), user (_) TO drive (_), user (_) - Correct (Y or N)?

TYPE the letter **Y** to accept.

OR

the letter **N** to exit the Replace Files option.

IF you typed the letter **Y** to accept, the following message is displayed on the screen:

Enter backup session name and touch RETURN

TYPE the name of the backup session that was written on the disk label when the files were backed up then press the RETURN key.

Result The system will load the program and then display the following message on the screen:

When disk to copy FROM is ready in drive (_) touch RETURN (or ESC to exit)

INSERT the backup disk marked number 1 in the disk drive, close the drive and then press the RETURN key.

Result The system will compute the files on the disk and display the file names and the following message on the screen:

When disk to copy TO is ready in drive (_) touch RETURN (or ESC to exit)

PRESS the RETURN key.

Result The screen will display each file name as they are being copied, the remaining amount of data to be copied, the amount of available space left on the disk being copied to.

WHEN all the informatiom from the backup disk has been copied, the following message is displayed on the screen:

```
00 More files left to replace.
When disk to copy FROM is ready in drive ( _ ) touch RETURN (or ESC to exit)
```

REMOVE the disk from the drive.

INSERT the disk marked number 2 in the disk drive and close the drive.

PRESS the RETURN key.

Result The following message will be displayed on the screen:

```
When disk to copy TO is ready in drive ( ) touch RETURN (or ESC to exit)
```

PRESS the RETURN KEY.

Result The screen will display each file name as they are being copied, the remaining amount of data to be copied, and the amount of available space left on the disk being copied to.

WHEN all of the files have been replaced, the following message is displayed on the screen:

```
Touch any key to exit. . .
```

Note: You would remove the backup disk and insert the CP/M system disk. Then press any key on the keyboard to return to the Disk Backup and Maintenance Menu.

IF there are more files to be copied, the following message will be displayed on the screen:

```
00 More files left to replace.
When disk to copy FROM is ready in drive ( _ ) touch RETURN (or ESC to exit)
```

REPEAT the steps above, starting with removing the disk.

Replace Files (Large files)

CHECK to be sure that the rigid disk partition that will be used, has enough space available before starting.

TYPE the number **3**

Result The following message is displayed on the screen:

```
The Xerox Disk Backup & Maintenance System
(C) 1982 Balcones Computer Corporation (P)
All Rights Reserved Version 0.00 (Month) 1982

REPLACE FILES

Note: Touch ESC any time to exit.

Enter Disk Drive that Files are to be copied FROM (A-P): ( _ )
```

TYPE the name of the disk drive that the files are to be copied from.

Result The following message is displayed on the screen:

```
Indicate User Area (00-31): ( _ )
```

TYPE the user area number that the files are to be copied from.

Result The following message is displayed on the screen:

```
Enter Disk Drive that Files are to be copied TO (A-P): ( _ )
```

TYPE the name of the disk drive that the files are to be copied to.

Result The following message is displayed on the screen:

```
Indicate User Area (00-31): ( _ )
```

TYPE the user area number that the files are to be copied to.

Result The following message is displayed on the screen:

```
Copy files FROM drive ( _ ), user ( _ ) TO drive ( _ ), user ( _ ) - Correct (Y or N)?
```

TYPE the letter **Y** to accept.

OR

the letter **N** to exit the Replace Files option.

IF you typed the letter **Y** to accept, the following message is displayed on the screen:

```
Enter backup session name and touch RETURN
```

TYPE the name of the backup session that was written on the disk label when the files were backed up then press the RETURN key.

Result The system will load the program and then display the following message on the screen:

```
When disk to copy FROM is ready in drive ( _ ) touch RETURN (or ESC to exit)
```

INSERT the backup disk marked number 1 in the disk drive and close the drive

PRESS the RETURN key.

Result The following message is displayed on the screen:

```
RECONSTRUCT FILE FROM PIECES

File (filename) . was split amoung several diskettes when it was backed up.

Place the Part 1 diskette in drive ( ) then type RETURN
```

PRESS the RETURN key.

Result The system will start copying part 1 of the file and display the following message on the screen:

```
r 000  Unsplitting file (filename)  .  Part 1
```

WAIT for the following message to be displayed on the screen:

```
Place the Part 2 diskette in drive ( ) then type  RETURN
```

REMOVE the disk marked part 1 from the disk drive.

INSERT the disk marked part 2 in the disk drive and close the drive.

PRESS the RETURN key.

Result The system will start copying part 2 of the file and display the following message on the screen:

```
000  Unsplitting file (filename)  .  Part 2
```

IF there are more parts of the file to be replaced, repeat the instructions above beginning with "WAIT for the following etc.".

WHEN all of the parts of the file that was split are replaced, the following message will be displayed on the screen:

```
Touch any key to exit . . .
```

REMOVE the backup disk and insert the CP/M system disk.

PRESS any key on the keyboard to return to the Disk Backup and Maintenance Menu.

Verify Disk Integrity

TYPE the number 4

Result The following message is displayed on the screen:

```
The Xerox Disk Backup & Maintenance System
(C) 1982 Balcones Computer Corporation (P)
All Rights Reserved Version 0.00 (Month) 1982

VERIFY DISK INTEGRITY

Verify which Disk: ( _ )
```

TYPE the name of the disk drive that you want to verify.

Result The Verify Disk Integrity option will start verifying the disk and display the following message:

```
00 Reading 0 to 0000 Blocks. (Touch ESC at any time to exit.)
```

WAIT for the following message to be displayed on the screen:

```
No errors detected. Touch any key to exit.
```

PRESS any key on the keyboard to return to the Disk Backup and Maintenance Menu.

Delete Files

TYPE the number **5**

Result The following message is displayed on the screen:

```
The Xerox Disk Backup & Maintenance System
(C) 1982 Balcones Computer Corporation (P)
All Rights Reserved Version 0.00 (Month) 1982

DELETE FILES

Delete Files on which Disk: ( _ )
```

TYPE the name of the disk drive that you want to delete files from.

Result The following message is displayed on the screen:

```
Indicate User Area (00-31): ( _ )
```

TYPE the number of the user area (00-30) that you want to delete files from.

Result The following message is displayed on the screen:

```
Erase Files on Drive ( _ ), User ( _ ) - Correct (Y or N)?
```

TYPE the letter **Y** to accept.

OR

the letter **N** to return to the Disk Backup and Maintenance menu.

Result The following message is displayed on the screen:

```
When Disk is ready in Drive ( _ ), touch any key to continue.
```

WHEN the disk to be deleted from is ready, press any key on the keyboard to continue.

Result The following message is displayed on the screen:

Enter the file name and touch RETURN:

TYPE the characters ***.*** to delete all of the files on the disk.

OR

the name of each file to delete, pressing the RETURN key once after each file name entered.

Result The following message is displayed on the screen with the names of the files that were entered:

Touch RETURN if you are finished entering file names.

WHEN you have entered the names of the files you want deleted, press the RETURN key.

Result The following message is displayed on the screen:

Type A to Delete all Files or Q to Query each File to Delete:

TYPE the letter **A** to delete all of the files that you've entered.

Result Each filename is automatically displayed on the screen as it is deleted.

OR

the letter **Q** to check (query) each filename before deleting each file.

Result Each file will be displayed on the screen and allow you to delete that file by typing Y or, leave that file on the disk and go to the next file by typing N. Repeat this procedure until all of the files that were entered are done.

WHEN you have finished deleting the files, the following message will be displayed on the screen:

```
All done. Type any key.
```

PRESS any key on the keyboard to return to the Disk Backup and Maintenance Menu.

Exit to CP/M

TYPE the number **6** to return to the CP/M operating system.

Result The following message is displayed on the screen for a few seconds, then the CP/M prompt is displayed:

```
The Xerox Disk Backup & Maintenance System
(C) 1982 Balcones Computer Corporation (P)
All Rights Reserved Version 0.00 (Month) 1982

EXIT TO CP/M
```

HOST TERMINAL

The Xerox 820-II Personal Computer has the ability to communicate with another host computer or time share system without using additional software. In the Host Terminal mode, the 820-II interfaces with a computer system as another terminal.

The Xerox 820-II can be remotely connected to the computer from any location that has access to a telephone by using dial-up modems or data sets. The Xerox 820-II Host Terminal operates in the full duplex mode.

You can use either the communication or printer port when entering the terminal mode to indicate which serial channel to use. If no port (parameter) is specified, the communication port will be used. To use the printer port, you would select B as the optional channel parameter (see directions below).

GETTING STARTED

In addition to the Xerox 820-II, you'll need the following equipment/items to use the Host Terminal application.

- A data-phone and telephone number - to access the host computer.
- An initialized double density working disk - which will be used if you want to save any data transferred from the host computer to the 820-II.

Note: Information that is scrolled off the top of the screen is saved in memory but not on disk. This information can be saved on a disk by using the save command shown on the next page.

When you turn the system ON or press the RESET button, the communication port is set for 300 baud, and the printer port for 1200 baud. Both serial ports are set for 7 data bits and even parity with 1 stop bit. When entering the terminal mode, the baud rate and serial port to be used may be specified. If no parameter is specified, the communication port set at 300 baud will be used. To use the printer port or change the baud rate, use the following procedures:

Selecting the Printer Port

Example: **TYPE** **B7 B** from the load page and press the RETURN key.

B = Baud command
7 = Baud rate (1200)
B = Printer port

Setting the Baud Rate

To change the baud rate at the communications port, you would first enter the baud command, then enter the baud rate code from the table below and press the RETURN key.

Example: **TYPE** **B7** and press the RETURN key.

B	=	Baud command
7	=	Baud rate (1200)
see note		Communication port

Note: If a port parameter is not specified, the communications port will be used.

BAUD RATE TABLE

0	hex	=	50	Baud
1	hex	=	75	Baud
2	hex	=	110	Baud
3	hex	=	134.5	Baud
4	hex	=	150	Baud
5	hex	=	300	Baud
6	hex	=	600	Baud
7	hex	=	1200	Baud
8	hex	=	1800	Baud
9	hex	=	2000	Baud
A	hex	=	2400	Baud
B	hex	=	3600	Baud
C	hex	=	4800	Baud
D	hex	=	7200	Baud
E	hex	=	9600	Baud
F	hex	=	19.2	Kbaud

Using the Host Terminal

To use the the Host Terminal application follow the instructions below. If you should make a mistake at any time, you may press the CTRL + ESC keys and start over.

USING the example and table above, set the baud rate (speed).

Note: The baud rate for the communications port is set at 300 baud when the system is turned ON or the RESET button is pressed.

TYPE the letter **H** and press the RETURN key to enter the Host Terminal mode.

Result The screen displays "Terminal mode. Touch CTRL+ESC to exit."

You can type and manipulate the information on the screen using the keys described on the next page.

If you are connected to the Host Computer with a Data Phone and Modem, proceed with the following instructions, **or refer to the instruction guide for your particular modem.**

PRESS	the TALK key on the Data Phone and dial the telephone number for the Host Computer.
Result	When the connection to the Host Computer has been made, you will hear an audible signal.
PRESS	the DATA key on the Data Phone.
Result	The link to the Host Computer is connected.
PRESS	the RETURN key.
Result:	The Host Computer will respond with: "PASSWORD"

You are now ready to sign on to the Host Computer. **Refer to your Host Computer manual for the proper sign on procedures.**

Keyboard Commands and Functions

This section describes the functions using the CTRL key and the numeric keypad keys when operating in the Host Terminal mode.

COMMAND	FUNCTION
CTRL+DOWN ARROW	Scrolls information down on the screen.
CTRL+UP ARROW	Scrolls information up on the screen.
CTRL+DEL	Enable local echo. Characters transmitted and received will be displayed on the screen.
CTRL+LINE FEED	Enable local auto line feed. When the RETURN key is touched, a line feed is also sent to the local screen but not transmitted.
CTRL+1	Enable remote echo. Characters received will be echoed back to the transmitting device. In this mode, the Xerox 820-II can act as a host to another terminal.
CTRL+2	Enable remote auto line feed. Carriage return characters received will be echoed to the remote device as carriage return and line feed codes.
CTRL+. (period)	Transmit BREAK. When these keys are touched a break condition will be enabled until CTRL+. is touched or any other character is typed.
CTRL+ESC	Exit the Host Terminal mode.

Saving Host Terminal Information

Text that is scrolled off the top of the screen when using the Host Terminal mode is stored in main memory. The information is stored in the Transient Program Area (TPA) of main memory and is large enough to store approximately 700 lines. It is stored in memory until power is turned off or an applications program is loaded into the Transient Program Area. This informatiom may be printed using the screen print function (CTRL+HELP) or stored on disk by loading the CP/M software and executing the SAVE command.

EXAMPLE: The command **SAVE 220 "filename"** will save the information in the CP/M file called "filename".

KILLESC

The KILLESC program allows the disabling of the CTRL+ESC command. The CTRL+ESC function is reactivated whenever the 820-II is reset.

To use the KILLESC program, you would enter the following command:

TYPE the word **KILLESC**, press the RETURN key and then follow the instructions on the screen.

Result The following message is displayed on the screen:

```
              KILLESC  version 0.00
         Copyright (C) 1982 XEROX Corporation

You may touch
              CTRL+C     to EXIT the program
              RETURN     to IGNORE the CTRL+ESC function or
              P          to PASS the CTRL+ESC sequence unaltered
```

SWAP

The SWAP program allows a physical disk drive to be referenced by another logical disk drive name. This utility allows taking advantage of the speed of the rigid disk drives with systems having both rigid disk drives and floppy disk drives.

When the system is first turned on, the floppy disk drives are always referred to as A through D, and the rigid disk drives as E through H. CP/M always requires a drive A for reset and "warm boot" operations. Warm boot and many applications programs are considerably faster from rigid disk drives than floppy disk drives, therefore it is desirable to use the speed of the rigid disk drives.

If CP/M is booted directly from one of the rigid drives (partition), then a "SWAP" command is done automatically. Loading CP/M from rigid drive H (partition H) would cause rigid drive H to be referenced automatically as the A Drive, and the original A Drive (floppy disk) would be referenced as the H Drive.

When using the SWAP program, each physical drive in the system can have its name changed. To change (swap) the names of disk drives B and F, you would enter the following command:

TYPE **SWAP b=f** and press the RETURN key.

Result The following message will be displayed on the screen:

```
Logical drive B: exchanged with F:
```

In this example the rigid drive (partition) normally called F will be accessed by CP/M whenever a program looks for information on the B Drive.

TIME AND DATE

The 820-II allows you to display the time and date on the screen.

To set the time and date:

TYPE the word **TIME** and press the RETURN key.

Result The following message is displayed on the screen:

Enter the date and time as: mm-dd-yy hh.mm.ss

ENTER your date as mm-dd-yy, your time in hours (hh) (expressed as zero to twenty three), minutes (mm) (expressed as zero to fifty-nine) and seconds (ss) (expressed as zero to fifty-nine).

PRESS the **RETURN** key to exit.

To display the date and time of day:

TYPE **TIME P** and press the RETURN key.

Result The time and date will be displayed on the screen as shown in the following example:

May 8, 1982 10:32:47

Once TIME is set, typing TIME M allows it to be reset.

SCREEN PRINT

The 820-II has the capability to print what is currently displayed on the screen.

To print what is displayed on the screen:

PRESS the **CTRL + HELP** keys.

Result The printer will print what is displayed on the screen.

Note: Do not change what is displayed on the screen until the printer has finished printing.

If you print a screen that is displaying graphics, the printed document will not be an exact duplicate of the screen.

SET

When you turn the system on, press the RESET button or load the CP/M software, the communication port is set for 300 baud and the printer port for 1200 baud. The SET utility allows you to change (set) the baud rate of the communication or printer port. To change the baud rate you would enter the SET command, the port you want to set, and then the baud rate as shown in the examples below.

To change (set) the baud rate (speed) for the printer port:

TYPE **SET PRINTER 600** and press the RETURN key.

Result The printer port is set for 600 baud.

To change (set) the baud rate (speed) for the communications port:

TYPE **SET COMM 9600** and press the RETURN key.

Result The communications port is set for 9600 baud.

SYSTEM COMPONENTS

TABLE OF CONTENTS

620 PRINTER (20 CPS)

INTRODUCTION

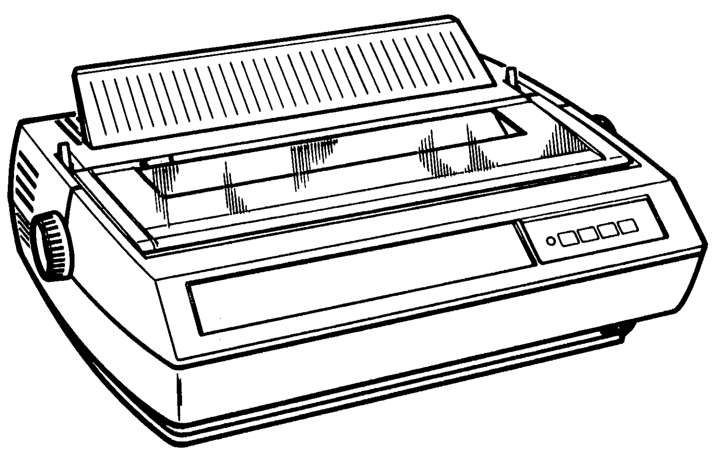
An illustration of the 620 printer (20 CPS) is shown below. If your printer does not match the illustration you may have a 630 printer (40 CPS), described on page 23.

The 620 Printer (20 CPS) is a serial printer designed for the low speed, low-to-moderate output requirements of standalone word and data processing business systems. The printer uses conventional data interchange techniques and protocol at speeds up to 1200 Baud.

The 620 Printer (20 CPS) produces a "typewriter" quality output of fully-formed characters at a maximum of 20 CPS. It includes operating features such as page formatting, graphics, positive and negative full and half line feed, absolute horizontal and vertical tabbing and proportional spacing.

The 620 Printer (20 CPS) features a new 98 character plastic printwheel with automatic recognition of printwheel type and language. A "drop in" printwheel exchange system is also featured where printwheels may be exchanged without removing the ribbon cartridge. The printwheels are available in many languages and type styles.

The 620 Printer (20 CPS) features quick change carbon film ribbon cartridges.



UNPACKING THE 620 PRINTER (20 CPS)

1. Inspect the outer carton and the various packing components as you open the carton. Note any damage which could have occurred during shipment.

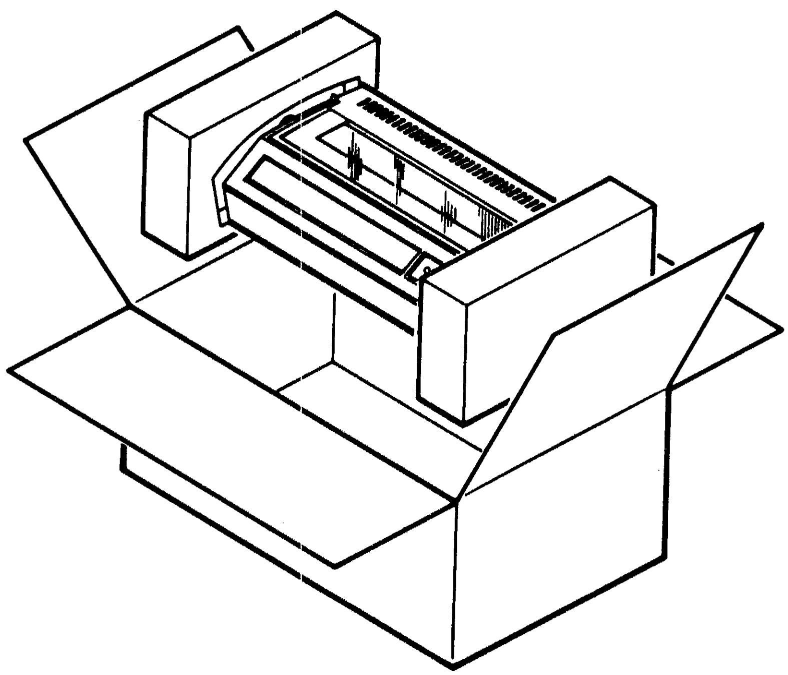
2. Remove any accessories packed around the printer.

3. Lift the printer out of the carton using the hand grips provided in the foam packing, and remove the foam packing end caps and plastic dust bag. Place the printer on a table or desk.

   NOTE: The weight of the printer is centered toward its right rear (as you look at it); thus it is easiest to lift by holding it at the right-rear and left-front.

4. Inspect the printer and any accompanying accessories for evidence of shipping damage. **Immediately notify the shipping agent of any damage to the unit or its parts.**

5. Remove any papers, tags or other materials found inside the printer, and remove any plastic tie or rubber band shipping restraints found.



INSTALLATION

1. Connect the interface cable to the connector at the rear of the printer and to the printer connector on the back of the screen.

2. Check the ON/OFF switch on the front of the printer -- it must be OFF!

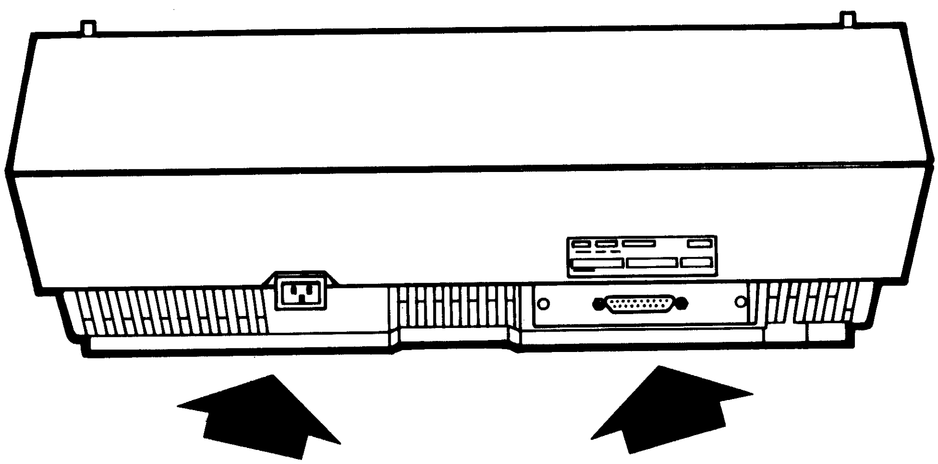
3. Plug the power cord into the outlet (left side, looking at the back of the printer) on the back of the printer.

4. Plug the other end of the power cord into the wall outlet. **DO NOT** turn on the printer yet.

5. If the platen has been removed, replace it (see instructions on page 17).

6. Install a printwheel and ribbon (see instructions on pages 9 and 12).

7. Check the switch settings (see instructions on page 18)



FAMILIARIZATION

1. **Control Panel** - Operator control switches and indicators.

2. **AC Power Switch** - This switch provides operator control of power to the printer.

3. **Front Access Cover** - Operator access to internal operating switches.

4. **Sound Panel** - Provides sound control when the printer is operating, and provides operator access for changing printwheels and ribbon cartridges and inserting paper.

5. **Paper Release Lever** - This lever releases paper roller pressure when pulled forward, allowing the paper to be positioned manually. Returning this lever to its back position re-establishes paper roller pressure.

6. **Paper Bail Lever** - The paper bail lever is used by the operator to move the paper bail forward to allow loading of paper, and back again to its operating position.

7. **Paper Rack** - This rack holds the paper up off the top rear of the printer to allow the flow of cooling air to exit the printer. Use of the paper rack is required in all cases where fan-fold (continuous) paper is used. Use of fan-fold paper without the paper rack will cause overheating of the printer.

   NOTE: Operation of the printer without the paper rack in such a mode will void warranty guarantees.

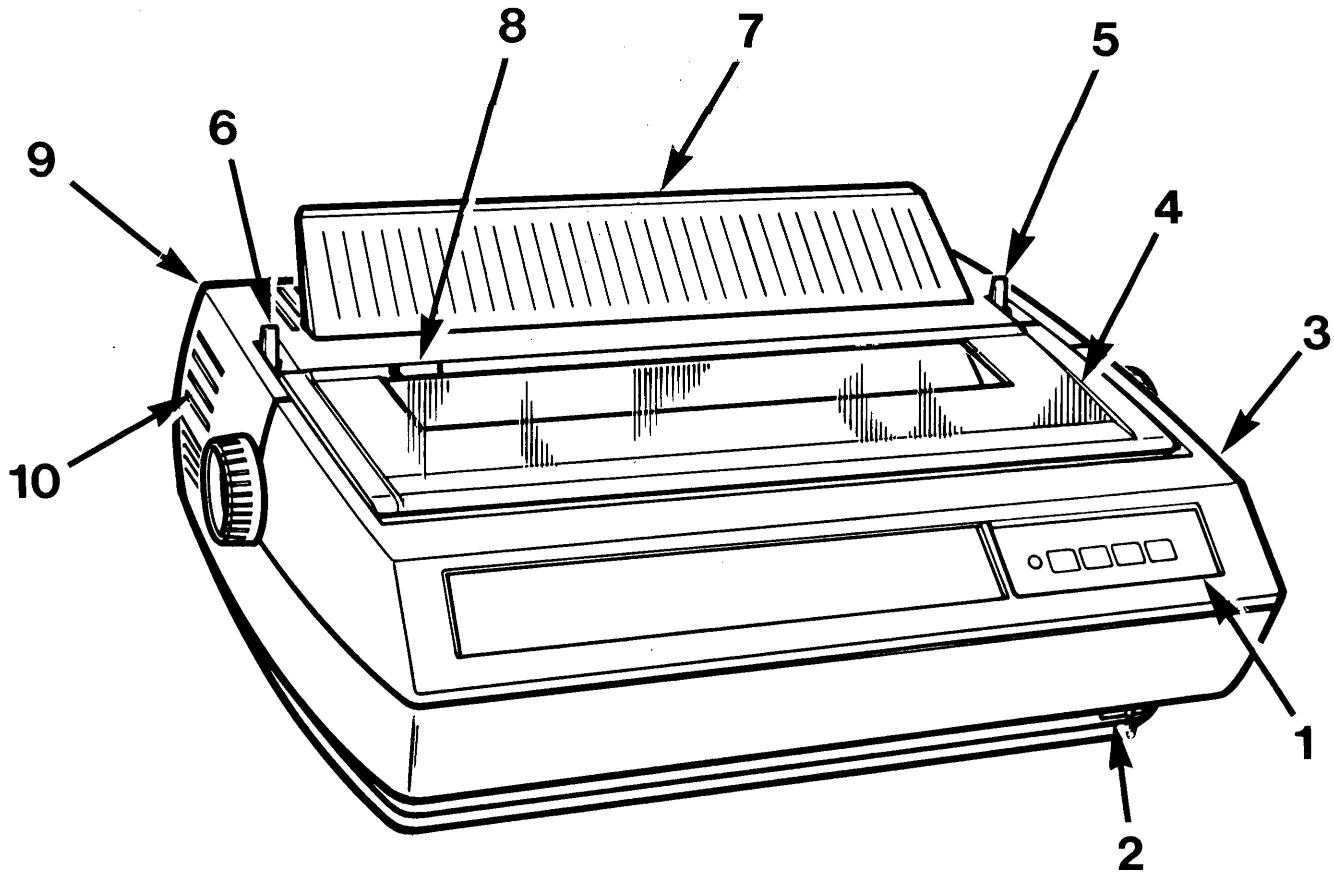
8. **Paper Guide** - A movable guide for aid in inserting paper.

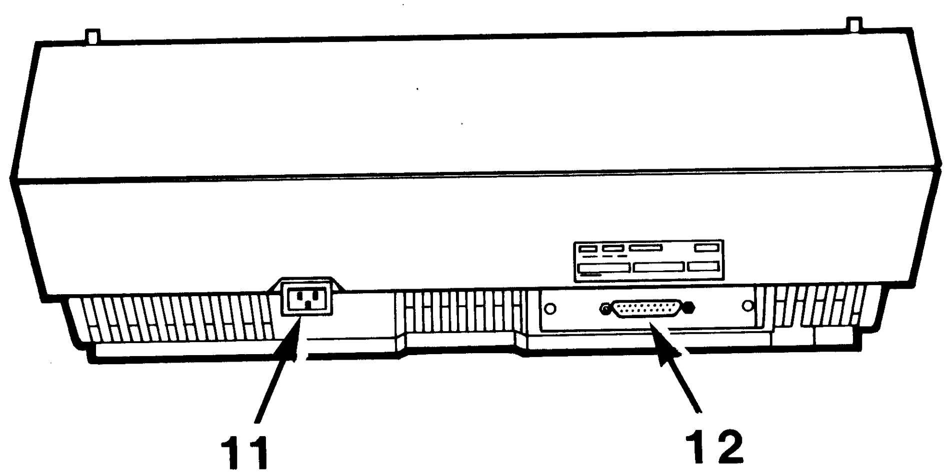
9. **Top Rear Cover** - The rear half of the top cover is easily removed whenever the printer needs service or maintenance.

10. **Electronics Compartment Ventilation** - These openings along with similar openings in the bottom provide for a flow of cooling air up through the electronics compartment.

11. **AC Power Input** - This plug on the right rear of the printer provides for connecting the AC power cable.

12. **Interface Connector** - This connector, located on the left rear of the printer provides for connecting the unit to a modem or host system.





SYSTEM COMPONENTS

Paper Handling Features

1. **Platen Knob** - This knob, when pushed in to disengage the drive gear, allows rotation of the platen to insert and position paper.

2. **Platen Release Latches** - These two latches are pressed down to release the platen for removal or when inserting a platen.

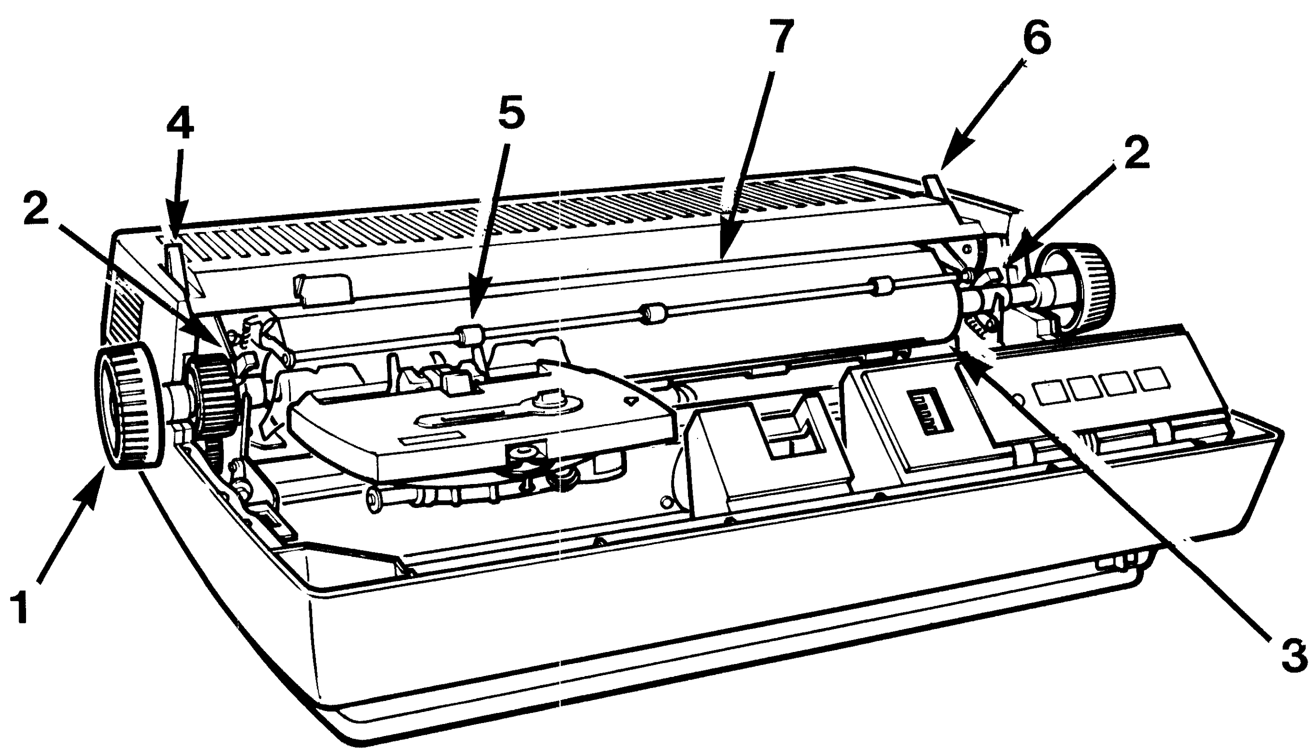
3. **Platen** - The platen is similar to those on standard office typewriters. The left-hand platen knob pushes in (to disengage the drive gear) for manual setting of paper position.

4. **Paper Bail Lever** - This lever is used to move the paper bail forward to allow loading of paper, and then back again to its operating position.

5. **Paper Bail** - The paper bail holds the paper against the platen for optimum print quality and quietness. The bail must be moved forward away from the platen using the paper bail lever when inserting paper.

6. **Paper Release Lever** - This lever releases paper roller pressure when pulled forward, allowing paper to be positioned manually. Returning this lever to its rearward position re-establishes paper roller pressure.

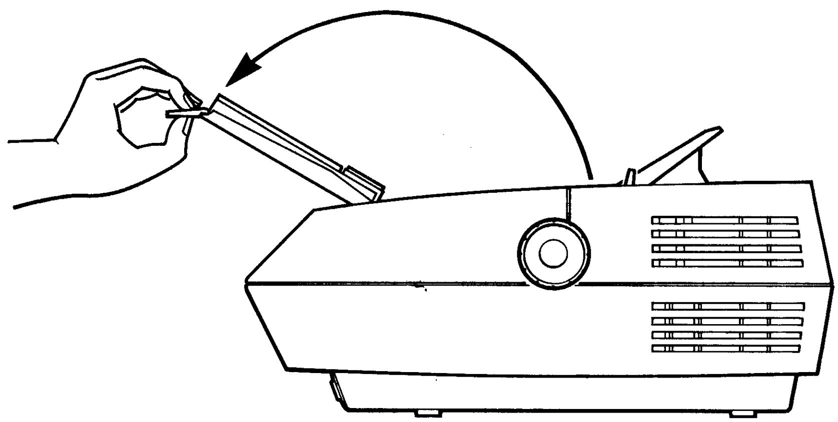
7. **Paper Scales Bar** - The scales on this bar aid in centering and spacing paper and copy. The bar itself helps direct the paper down behind the platen during insertion.



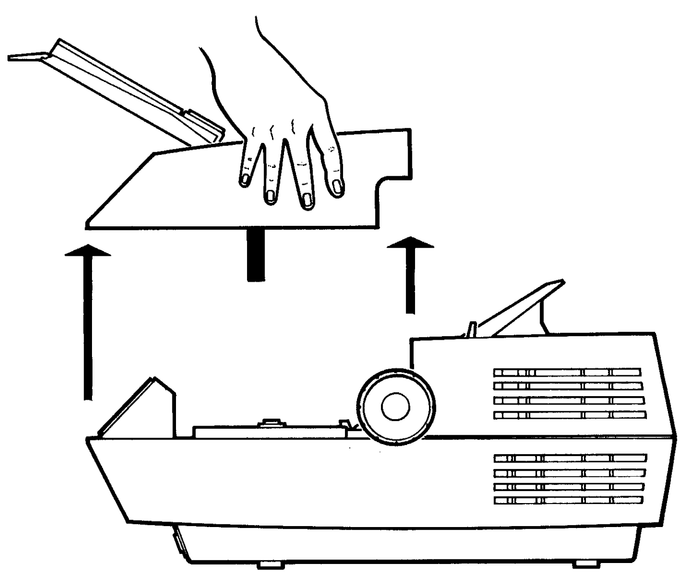
PREPARATION FOR OPERATION

Remove Access Cover

1. Open the sound panel by grasping the edge towards the back of the printer and lifting it towards the front of the printer.



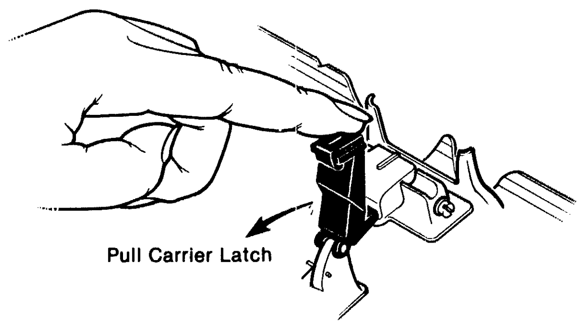
2. Press both latches on the inside of the front access cover and remove it by lifting it from the printer.



Removing a Printwheel

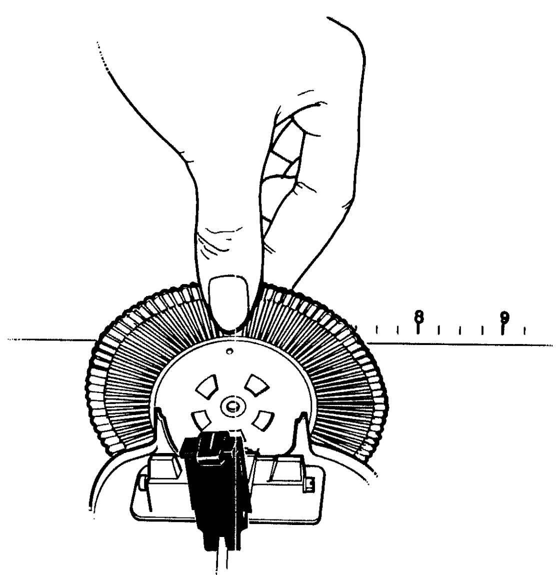
1. Remove the printer cover.

2. Release the CARRIER LATCH by pulling it toward you with your finger. It will snap open, so the printwheel can be easily removed. (It is not necessary to remove the ribbon.)



3. Slide your finger between printwheel and ribbon.

4. Grasp the top of the printwheel with your thumb and first finger and lift it straight up.

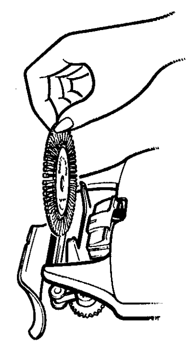


Installing a Printwheel

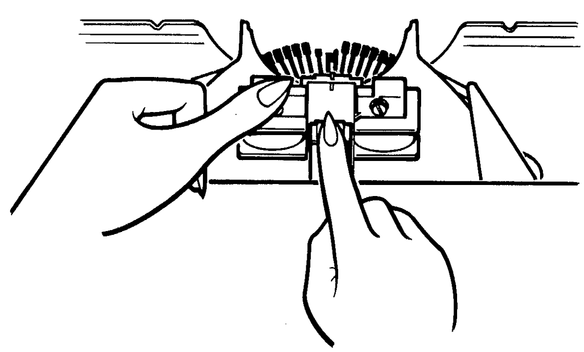
1. Be sure CARRIER LATCH is open.

2. Grasp the printwheel gently between your thumb and first finger. Be sure the pitch printed on the pointer is toward you.

3. Lower the printwheel into the carrier (between the ribbon/correction tape and the ribbon guide).



4. Push the carrier forward and hold it in place while you push the Carrier Latch forward into its groove.



5. When you touch any character, the printwheel spins as the printer locks it into the correct position for printing.

If the printwheel continues to spin and the printer beeps twice, it means that it can't identify the printwheel. Open the Carrier Latch to verify that the printwheel is inserted correctly. Then close it again and touch any key. If it still spins and beeps, try another printwheel.

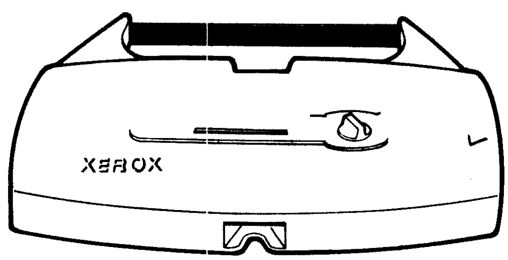
The Ribbon Cartridge

Before you install a new ribbon cartridge in the printer notice:

1. The WINDOW lets you see how much ribbon remains in the cartridge. The ribbon feeds from the left side of the cartridge to the right side, so that a new cartridge should have all of the ribbon showing on the left side.

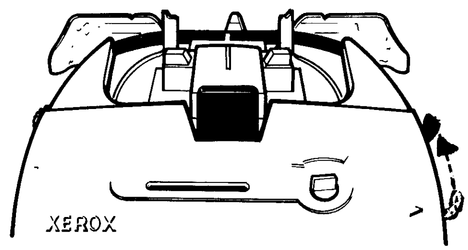
2. The colored KNOB tightens the ribbon across the cartridge guides. You turn it counterclockwise to tighten the ribbon (look at the arrow above the knob). You cannot tighten the ribbon when the cartridge is locked in the printer.

3. At the back of the cartridge is a GUIDE SLOT. This slot helps you center the cartridge correctly as you install it.

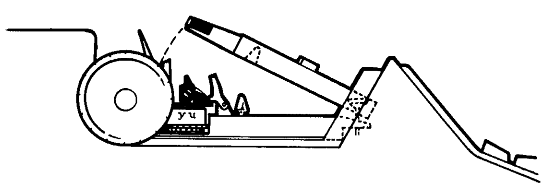


Removing a Ribbon Cartridge

1. Raise the sound panel.
2. Release the CARRIER LATCH as if you were about to remove the printwheel (see printwheel removal instructions).
3. Push the red CARTRIDGE RELEASE LEVER (under the right side of the ribbon) toward the platen to the OPEN position.



4. Hold the sides of the cartridge with both hands.
5. Tilt the front of the cartridge upward, then lift the cartridge up and out of the printer.

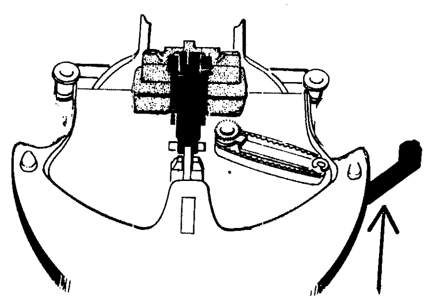


Installing a Ribbon Cartridge

1. Take up any tension in the ribbon by turning the colored knob on the top of the cartridge in the direction of the arrow.

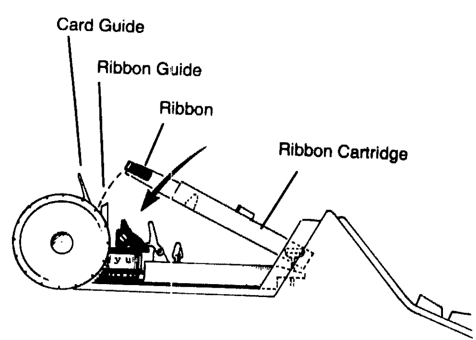
2. Move the red CARTRIDGE RELEASE LEVER at the side of the carrier to the open position.

3. Return the CARRIER LATCH if it is not released.



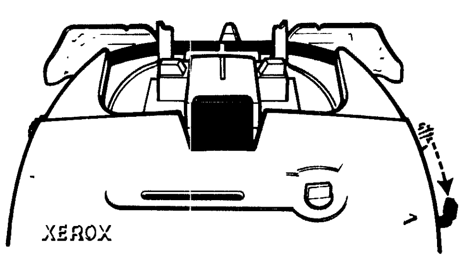
4. Hold the cartridge in both hands, tilt the back of the cartridge down, and slide it backward into the rear guide as you lower it onto the carrier (see illustration below).

5. As you lower the cartridge, position the ribbon between the plastic card guide and the ribbon guides until the cartridge fits securely on the carrier.



6. Slowly move the CARTRIDGE RELEASE LEVER to its locked position.

7. Push the carrier forward and hold it in place while you push the CARRIER LATCH forward into its groove.



Inserting Paper or Forms

Inserting paper or forms into the printer is accomplished in much the same manner as in a standard typewriter. Paper is inserted down behind the metal paper out bail and platen while the platen is turned manually to bring the paper around and up in front of the platen. The front paper bail aids in guiding the paper back over the platen to the rear when pulled forward. The paper release lever at the right-hand side of the printer may be pulled forward to release roller pressure after paper insertion so the paper can be properly aligned in the printer. After paper is positioned, both the front paper bail and paper release lever are returned to their operating positions.

OPERATING THE 620 PRINTER (20 CPS)

Preliminary Steps

1. Install a printwheel, a ribbon cartridge and insert a sheet of paper.

2. Check switches as shown on page 18 of this section.

3. Move the carriage manually to the right a short distance.

4. Move the power ON/OFF switch at the right front of the printer to ON.

5. The POWER indicator should glow; the carriage should move to the left slowly, and then back to the right, to stop at the first print position; and the printwheel should rotate and stop at its "home" position (i.e., the "flag"). This entire process is called the INITIALIZATION, RESET or RESTORE sequence. It clears all volatile memory, resets all position counters and sets the printer to print the first character.

SPECIAL CONSIDERATIONS

1. **Cleaning Printwheels**

   Printwheels used with carbon ribbons seldom need cleaning. Printwheels used with cloth ribbons will require an occasional cleaning.

   Remove the printwheel and clean with toluene or naphtha* and a soft brush or wiper. DO NOT clean with water. Make sure to keep the reflective "code" segments on the back or character side of the printwheel clean and shiny. Be careful not to bend the "spokes".

2. **Cleaning the Platen and Paper Rollers**

   Platens and paper rollers are made of rubber. They require periodic cleaning for a more positive friction paper drive. Periodically clean the platen, paper bail rollers and pressure rollers with soft tissues or cloth wipers and a good commerically available platen cleaner.

3. **Cleaning the Card Guide**

   Remove the ribbon cartridge, printwheel and platen. Remove the card guide and clean it using soft tissue or a soft cloth and alcohol.

CAUTION: DO NOT use alcohol, water or platen cleaners to clean printwheels.

DO NOT use alcohol to clean platens or other rubber parts (it hardens the surface).

DO NOT use platen cleaner to clean the card guide or other plastic parts (it attacks the plastic).

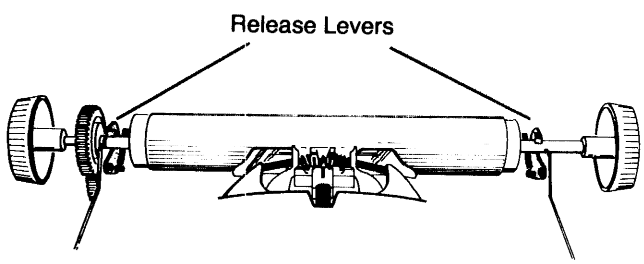
* Toluene and naphtha are available at most drug stores.

For safety, unplug your printer before you begin to clean it. Avoid spraying or pouring liquids directly on the printer.

Special cleaning materials are available from Xerox.

Cleaning the Platen and Card Guide.

1. Remove the printer cover.
2. Lift the paper bail, and the tissue deflector.
3. Release the carrier latch and remove the printwheel and ribbon.
4. Push back on the platen release levers at each end of the platen.
5. Lift the platen up and out.

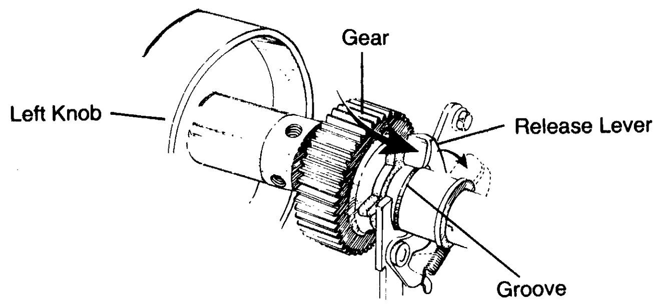


6. Clean the platen.
7. Clean both sides of the card guide.

8. To replace the platen:

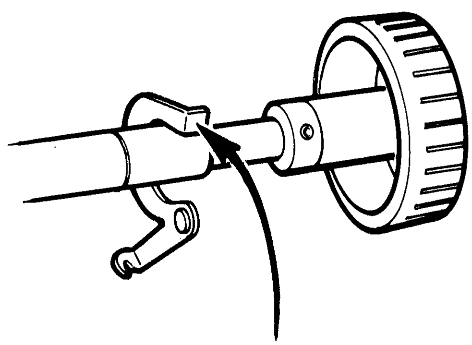
   a. Hold the platen over the release levers.

   b. Lower the left side of the platen and push the left release lever back with the metal platen rod until the groove falls into the guide.



   c. Lower the middle of the platen behind the plastic paper guide.

   d. Push the right release lever back with your right thumb far enough to lower the metal platen rod into the right guide. The release lever will snap over the rod when you have placed the rod in the correct position.



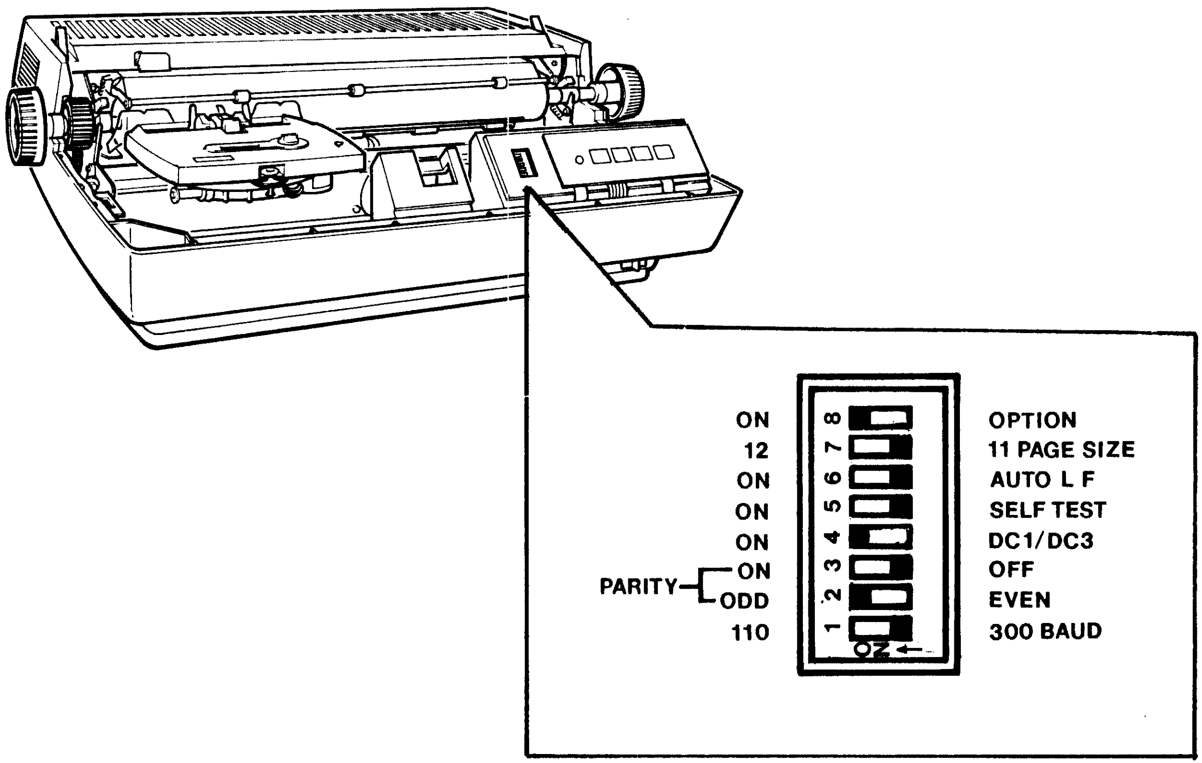
9. Reinstall the printwheel and ribbon.

10. Lower the tissue deflector, paper bail and typewriter cover.

Setting the Switches Under the Front Cover

The printer has been factory preset to the proper switch positions for use with a Xerox 820-II Personal Computer.

These switches are located to the left of the external Control Panel, but is covered when the front cover is in place. These switches control operating modes and ordinarily do not require attention once set.



OPTION: This switch when ON, enables the printer to receive and transmit data at a speed of 1200 Baud.

PAGE SIZE: This switch enables setting page size, used in the Top Of Form/Form Feed function, to either the US standard 11" or the European standard 12" page length.

AUTO LF: When ON, this switch enables the 20 CPS to automatically advance the paper one line with each carriage return. This relieves the host system of the need to send a line feed command with each carriage return command.

SELF TEST: If this switch is in the ON position when the printer is turned on, the printer will enter a self test mode and begin sequencing thru its self test program. The Control Panel PAUSE and RESET switches may be used to interrupt the self test sequence. To exit the mode, the SELF TEST switch must be moved to OFF and the power to the 20 CPS must be turned off momentarily.

DC1/DC3: This switch is used to allow the printer to operate with much faster host systems without loss of data. When ON, special characters (DC1/DC3) are transmitted between the printer and the host automatically whenever the print buffer is either nearly full or nearly empty.

Note: The 820-II does not support ETX/ACK operation.

PARITY ON-OFF: This switch enables parity checking and parity information transmission when on.

PARITY ODD-EVEN: This is used in conjunction with Parity ON-OFF to determine the nature of parity information handling.

110-300 BAUD: This switch selects 110 or 300 Baud as the speed at which the printer will receive and transmit data. If 1200 Baud is selected (OPTION switch ON), this switch doesn't have any affect on printer operation.

For use with the Xerox 820-II, all the switches should be positioned to the right of the printer except for switches 2, 4 and 8. They should be positioned toward the left of the printer.

Using The Control Panel Switches

These five switches are located to the right of the Control Panel where they are accessible to the operator with all covers on the machine. These are membrance type momentary action switches actuated by a touch of the finger.

RESET: This switch will clear an "error" indication and return the printer to operation. It will also return the printer to operation following a PAUSE command.

PAUSE: Touching this switch will cause the printer to stop printing without any loss of data, and the power indicator will go out. Printing is continued by pressing the RESET switch.

LINE FEED: Touching this switch initiates a single line feed. Action is repeated if the switch is held activated longer than 1/2 second. A line feed code will not be transmitted.

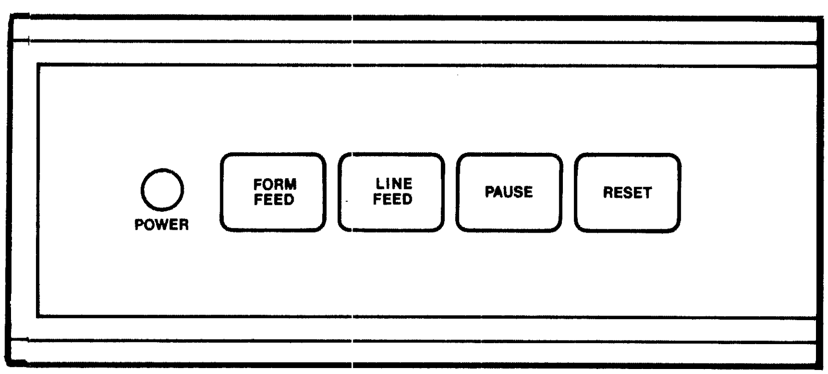
FORM FEED: Touching this switch initiates a form feed to the next top-of-form position. A form feed code is not transmitted.

The POWER Indicator

The power indicator glows whenever power is turned on to the printer.

The indicator will flash for the following conditions:

1. A parity error was detected with the PARITY switch on.
2. The printer buffer (memory) has overflowed.
3. The printer didn't receive a "Data Set Ready" signal.



SPECIFICATIONS

Print Speed: Up to 20 characters per second.

Character Set: 98 printable character printwheel.

Printwheels: Plastic 98 character Xerox.

Character Spacing: 10-pitch = 10 characters/inch
12-pitch = 12 characters/inch
15-pitch = 15 characters/inch

Column Spacing: 1/120 inch (.21mm) minimum.

Print Line: 13.2 inches (335.3mm)
132 columns 10 pitch

Paper Width: 13.2 inches (387.4mm) maximum - friction feed platen.

Carriage Speed: 1.7 sec maximum for 13.2 inches (332.77mm) of motion.

Tabulation: Left or right.

Line Spacing: 1/48 inch (.53mm) minimum.

Paper Feed: Bidirectional.

Paper Thickness: 1 to 5 part forms; maximum overall thickness .024" (.61mm).

Other Features: Self test; host program control through escape sequences; data receive/transmit speed selection.

Power Requirements: Operation from nominal 120/220-240 volt AC inputs, 50-60 Hz. 120W maximum power consumption.

CHECK YOUR PRINTERS's SERIAL PLATE FOR PROPER INPUT POWER.

630 PRINTER (40 CPS)

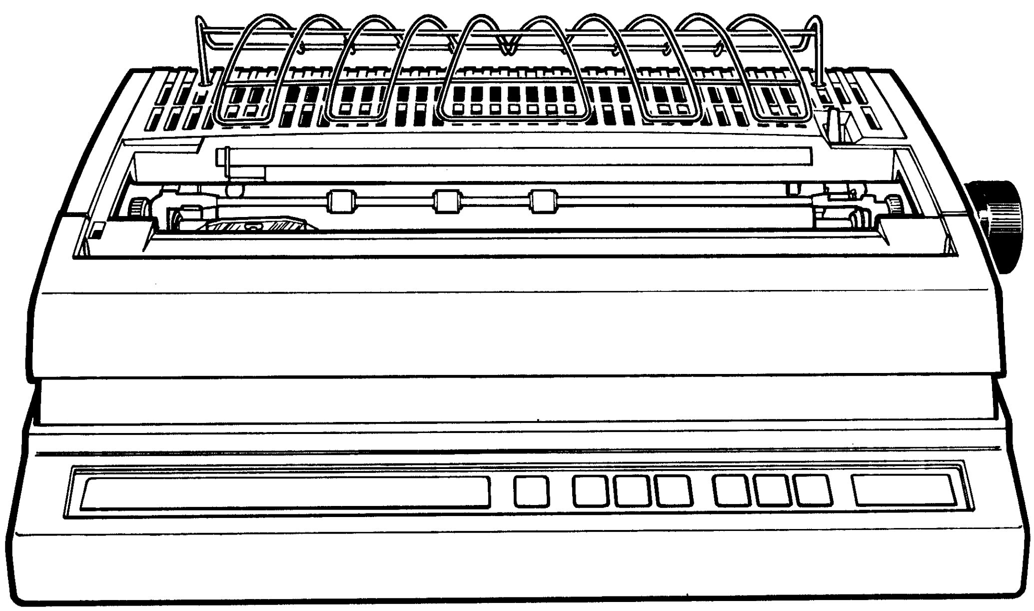
INTRODUCTION

An illustration of the 630 printer (40 CPS) is shown below. If your printer does not match the illustration you may have a 620 printer (20 CPS), described on page 1.

The 630 Printer (40 CPS) is a universal RS 232-C interface printer. It will support a heavy workload using conventional serial data interchange techniques and protocols.

The 630 Printer (40 CPS) can use all Diablo and Xerox plastic and metal printwheels.

The 630 Printer (40 CPS) can use many of Diablo's present paper handling devices, such as forms tractors, sheet feeders, etc.



UNPACKING THE 630 PRINTER (40 CPS)

1. Take the printer and all accessories out of the carton and remove the plastic dust bag. Place the printer on a desk or table.

   NOTE: The weight of the 630 Printer (40 CPS) is centered toward its right-rear (as you look at it); thus it is easiest to lift by holding it at the right-rear and left-front.

2. Inspect the printer and any accompanying accessories for evidence of shipping damage. **Immediately notify the shipping agent of any damage to the unit or its parts.**

3. Remove the access cover. It is held in place by magnetic latches in front and small tabs in back.

4. Remove the plastic bag and the CAUTION tag attached to the paper bail by cutting or removing the tie wrap (nylon strap).

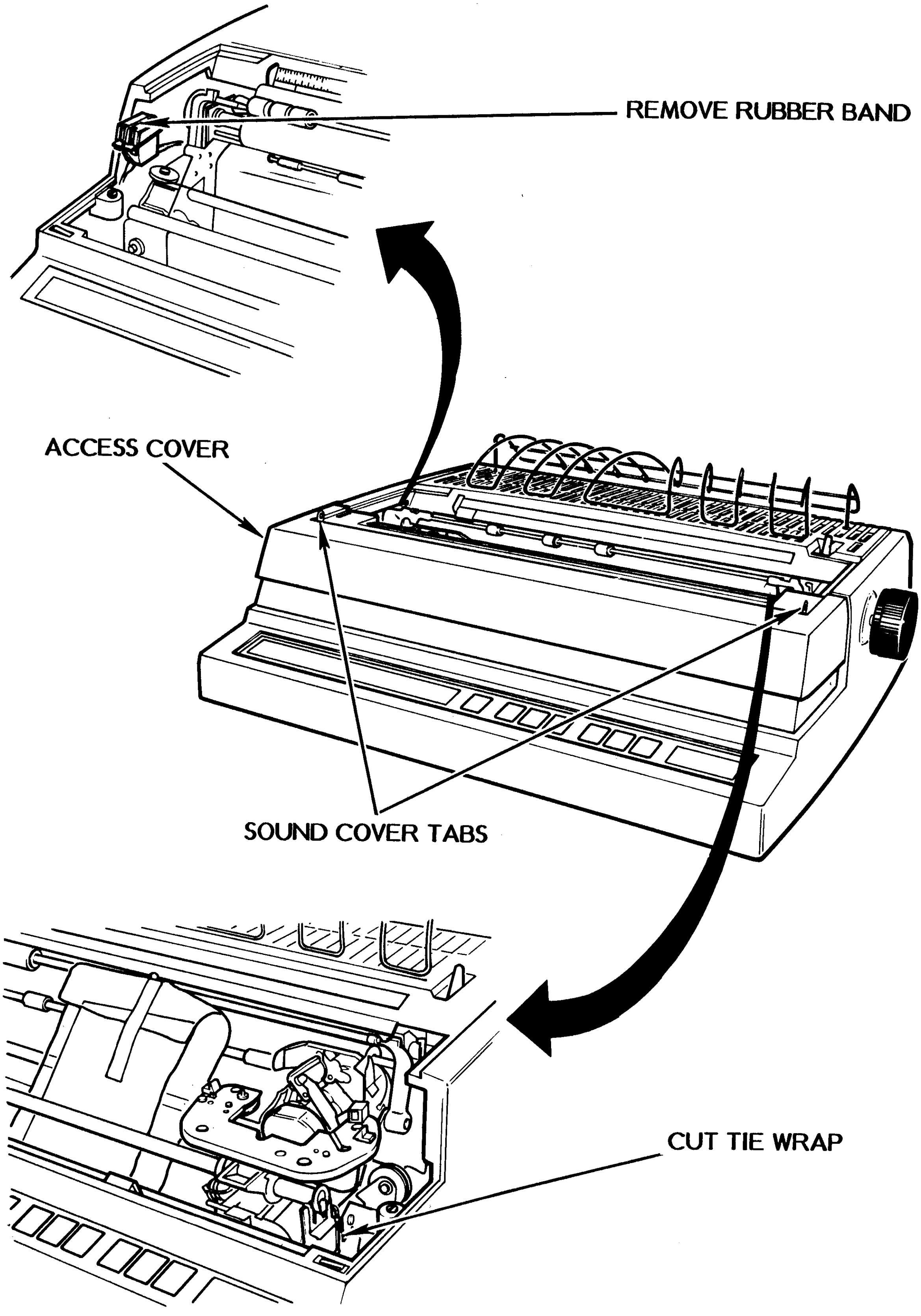
5. Remove or cut the following shipping restraint items, if installed (see pictures on opposite page):

   a) Rubber band securing the cover open switch actuator.

   b) Rubber band securing the paper cradle to the paper pressure rollers (if no platen is installed).

   c) Tie wrap (nylon strap) holding the carriage to the right printer frame.

   CAUTION: DO NOT cut any other tie wraps!



REMOVING SHIPPING RESTRAINTS

INSTALLATION

1. Place the 630 Printer (40 CPS) on your desk or table. Look at the two cords. One printer cord plug fits the large outlet on the back of the printer; the other plug fits the large outlet on the back of the screen. One power plug fits the smaller outlet on the back of the printer; the other plug fits a wall outlet.

2. Check the ON/OFF switch on the back of the printer -- it must be OFF!

3. Plug the printer cord into the large outlet (right side, looking at back -- see picture on the opposite page) on the back of the printer. Tighten the screws on the plug.

4. Plug the L-shaped end of the power cord into the small outlet (left side, looking at back -- see picture on the opposite page) on the back of the printer.

   Plug the other end of the power cord into the wall outlet. **DO NOT** turn on the printer yet.

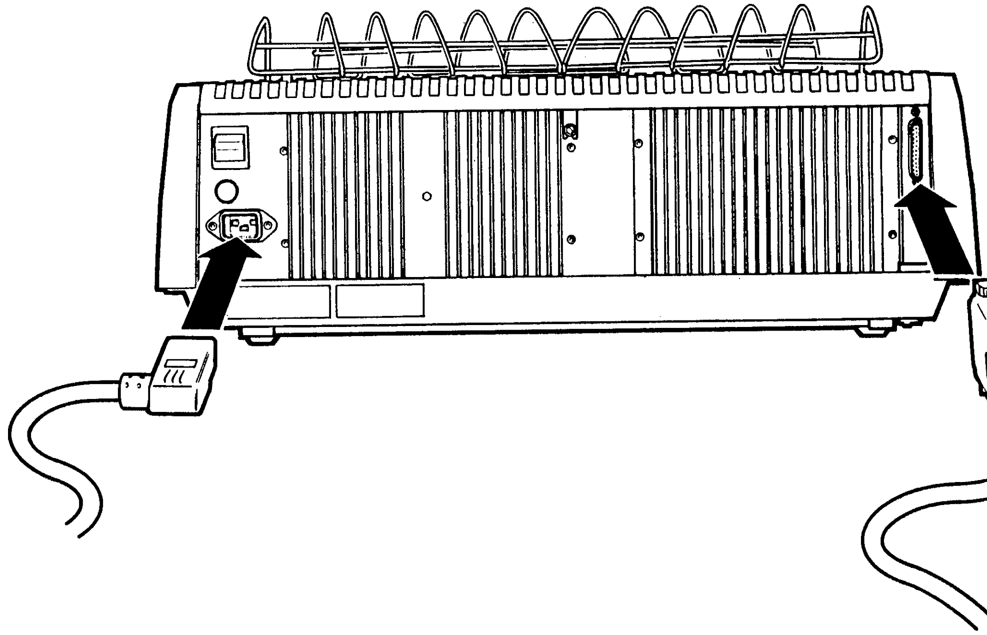
5. If the platen has been removed, reinstall it by lowering it down into place while pressing down on both platen release levers. (See item number 3 on page 29 for an illustration.)

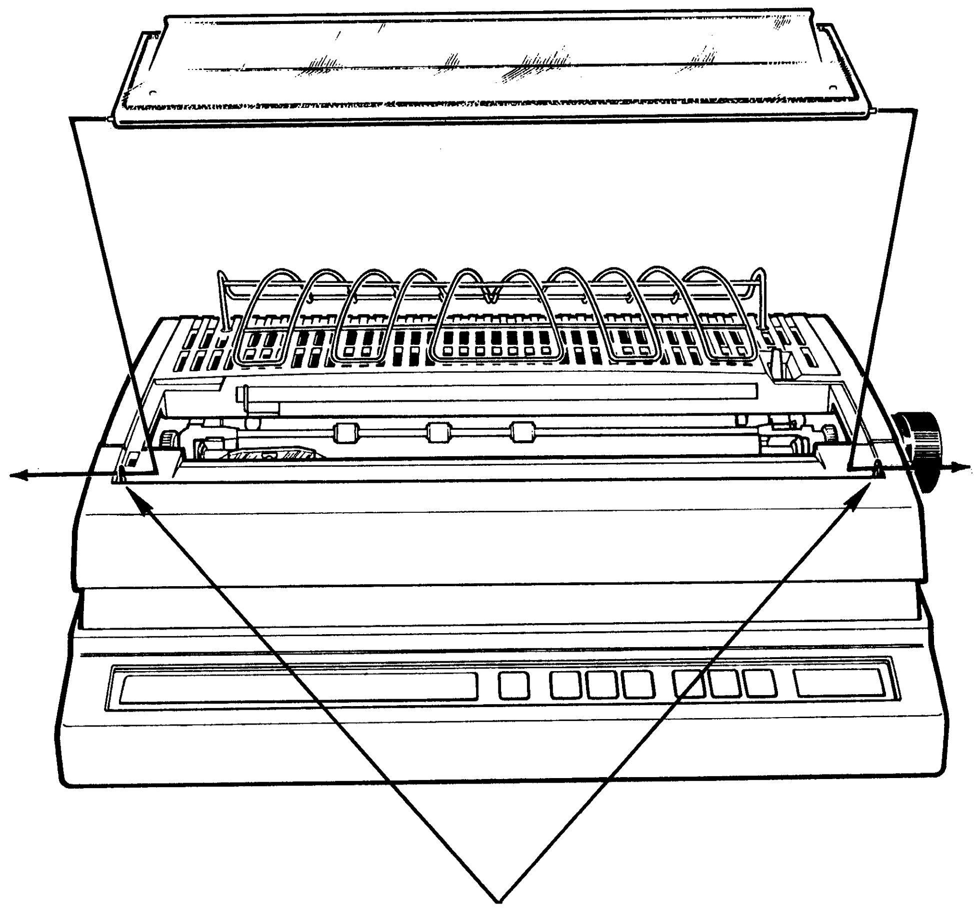
   NOTE: Make sure the platen's releasable drive gear end is on the right (as you view it).

6. Install the sound shield on the front access cover as shown in the illustration on the bottom of page 27.

7. Install the platen knob on the right end of the platen - thru the hole in the top cover. Engage the knob on the platen shaft, rotate the knob until its slot fits over the cross pin on the platen shaft, then push against the knob to snap it into place.

8. Now you are ready to read the information on page 28 and then install a printwheel and ribbon using the instructions on pages 30 and 31.





SOUND COVER TABS

FAMILIARIZATION

(See picture on opposite page)

1. **Control Panel** - Operator control switches and indicators.

2. **Access Cover** - Operator access to internal controls and for changing printwheels and ribbons.

3. **Platen** - Similar to those on standard office typewriters. There is a hand knob on the right side only.

Paper Handling Features

4. **Platen Knob** - This knob, when pushed in (to disengage the drive gear), allows the operator to rotate the platen to insert and position paper.

   Platen Release Latches - These two latches are pressed down simultaneously to insert a platen or to release the platen for removal.

5. **Paper Bail** - The paper bail holds the paper against the platen for optimum print quality and quietness. The bail must be pulled forward (away from the platen) when inserting paper.

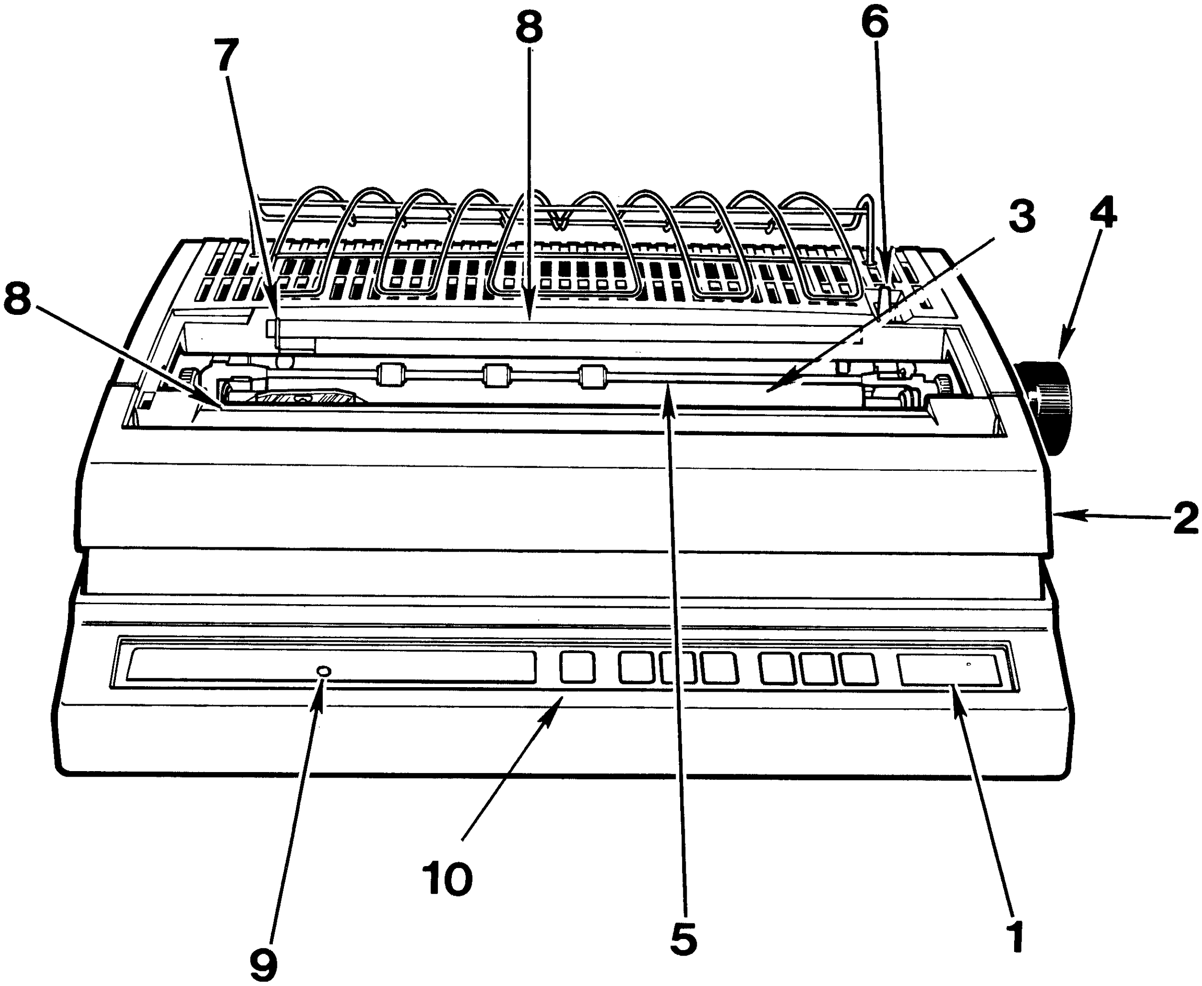
6. **Paper Release Lever** - This lever releases paper roller pressure when pulled forward, allowing the paper to be positioned manually. Returning this lever to its back position re-establishes paper roller pressure.

7. **Paper Guide** - A movable guide for aid in inserting paper.

8. **Paper Scales** - There are two scales associated with the printer. One is a column indicator, located on the top cover; the second is a paper scale mounted on the access cover. These scales aid the operator in centering and spacing paper and copy.

9. **Power Indicator Light** - This light indicates that the power is on.

10. **Reset Button** - This button must be pressed each time the cover has been opened.


8
7
6
3
4
8
2
10
9
5
1

PREPARATION FOR OPERATION

Installing a Printwheel

CAUTION: Ensure that power to the 630 Printer (40 CPS) is turned OFF!

1. Remove the access cover, if its in place.

2. Grasp the print hammer assembly (the cylinder with a red line on top of it) and pull it toward you to tilt the printwheel mechanism away from the platen and card guide.

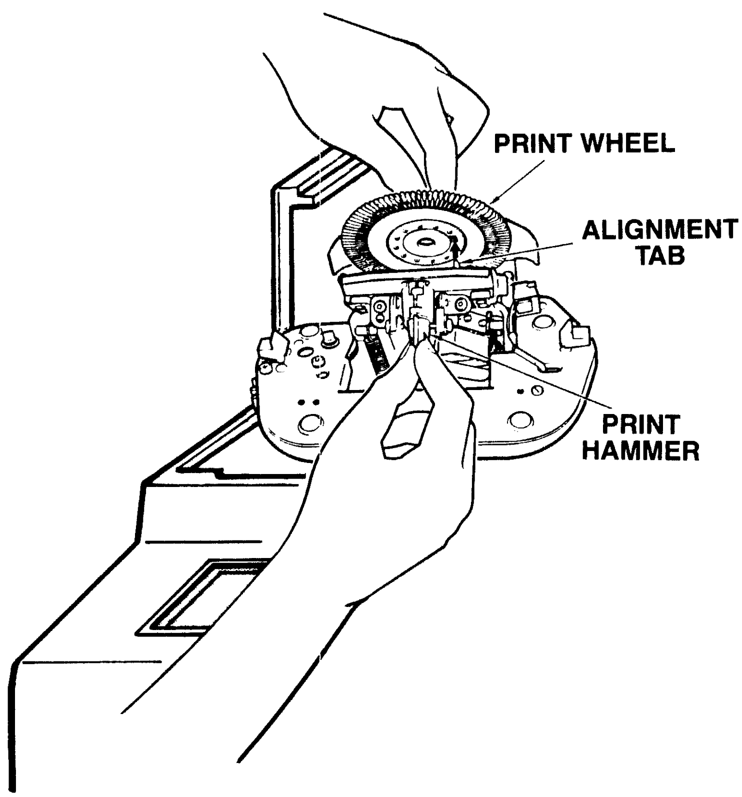
3. Grasp the printwheel (metal or plastic) by its rubber hub and place it on the printwheel motor hub. Align the wheel's alignment slot with the hub's alignment tab, and push the wheel firmly to fully seat it on the motor hub.

4. Tilt the printwheel mechanism back to its operating position.

Removal of the printwheel is simple. Tilt the printwheel mechanism toward you, grasp the printwheel by its rubber hub and pull it free of the printwheel hub.

Note: Printwheels are rugged and dependable, but they can be damaged. Use care when handling them to avoid bending the "**spokes**". Always store printwheels in their plastic containers when they are not installed in the printer.

6. Install a ribbon using the instructions on the next page.



Installing a Ribbon Cartridge

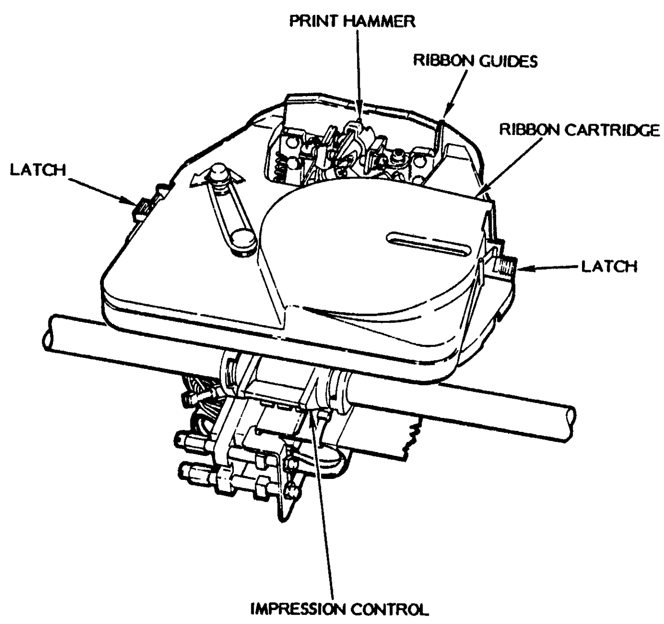
1. Open the plastic envelope and take out the ribbon cartridge. Note the small knob on the top surface of the cartridge for advancing the ribbon manually. Use this knob to take up any slack in the exposed portion of the ribbon and to make sure the ribbon is tight and straight.

2. Hold the cartridge in one hand with the exposed ribbon toward the platen. Lower the cartridge down over the print hammer guide (orange stripe). Be sure the exposed ribbon is straight and located between the card guide and print-wheel. Push the cartridge down firmly until both latches have snapped into position. Rock the cartridge back and forth on the platform to ensure that the ribbon is free to move up and down. Turn the ribbon advance knob a turn or two to ensure that the ribbon is tight, straight and ready to operate.

   The ribbon cartridge may be removed by pressing down on both latches simultaneously. The cartridge will be raised up slightly and may be grasped easily and lifted out of the printer.

   Note: When a ribbon cartridge is nearly empty, a yellow cross-hatched pattern will appear on the visible back side of the ribbon. The printer will stop printing, sound its alarm and the RIBBON/PAPER light will come on if printing is attempted with the ribbon in the warning (yellow) zone.

3. Check to be sure the switches (located in front of the ribbon and behind the control panel) are set as indicated on page 35.



Inserting Paper or Forms

Inserting paper or forms into the 630 Printer (40 CPS) is accomplished in much the same manner as in a standard typewriter. Paper is inserted down behind the metal paper out bail and platen while the platen is turned manually to bring the paper around and up in front of the platen. The front paper bail aids in guiding the paper back over the platen to the rear when pulled forward. The paper release lever at the right-hand side of the printer may be pulled forward to release roller pressure after paper insertion so the paper can be properly aligned in the printer. After paper is positioned, both the front paper bail and paper release lever are returned to their operating positions.

Press the RESET switch on the front of the printer.

Paper Thickness/Print Intensity Adjustment

The two-position **Multicopy** lever located at the front of the carriage assembly adjusts for paper thickness and print intensity. Setting the lever to its upper position moves the carriage close to the platen, and actuates a switch to the proper setting for light and medium weight paper and form sets of up to two carbon copies. For heavier paper or form sets of up to five copies, the **Multicopy** lever is set to its lower position. This rocks the carriage away from the platen slightly, and deactivates the switch to enable an increased print intensity.

To avoid the possibility of ribbon damage, the **Multicopy** lever should always be set at its upper position when printing on single sheets of paper using carbon ribbons.

OPERATING THE 630 PRINTER (40 CPS)

Preliminary Steps

1. Install a printwheel, a ribbon cartridge and insert a sheet of paper.

2. Move the carriage manually to the right a short distance.

3. Move the power ON/OFF switch at the right rear of the printer to ON.

The POWER indicator should glow; the carriage should move to the left slowly, and then back to the right, to stop at the first print position; and the printwheel should rotate and stop at its "home" position (i.e., the "flag" on metal printwheels should be at the top if the Printwheel Select switch - under the access cover - has been properly set). This entire process is called the INITIALIZATION, RESET or RESTORE sequence. It clears all volatile memory, resets all position counters and sets the printer to print the first character.

Paper Handling Accessories

Forms Tractor/Pin Feed Platen: These devices facilitate precision handling of the continuous or manifold paper forms and are provided in both unidirectional and bidirectional versions.

Mechanical Front Feeder: This device also mounts on top of the 630 Printer (40 CPS). It is intended for use with difficult, multipart forms and the heavier ledger card stocks.

Bottom Feed: The can be configured for feeding continuous or manifold forms up through the bottom of the machine. It must be used with either a pin feed platen or a forms tractor.

SPECIAL CONSIDERATIONS

1. Cleaning Printwheels

Printwheels used with carbon ribbons seldom need cleaning. Both plastic printwheels and metal printwheels used with cloth ribbons will require an occasional cleaning.

Remove the printwheel and clean with toluene or naphtha* and a soft brush or wiper. DO NOT clean with water. DO NOT get solvent on the hub or damper ring (metal wheels). Be careful not to bend the "spokes".

2. Changing Ribbons During Operation

The 40 CPS printer will stop printing, the RIBBON/PAPER light will appear, and the alarm will sound upon reaching the end of a carbon ribbon. Should this happen during receipt and print out of data from a host system, the operator should open the access cover, replace the ribbon cartridge as described, close the access cover, and then touch the control panel RESET switch to resume printing.

3. Cleaning The Print Hammer

Remove the printwheel and ribbon cartridge as described. Locate the movable print hammer inside its guide and push the hammer out to the rear, to expose as much of the head as possible. Use toluene or naphtha* and a wiper or brush to remove any accumulated ink or other substances, and wipe dry.

* Toluene and naphtha are available at most drug stores.
* 90% isopropyl alcohol may be used as a solvent for cleaning the print hammer.

Setting The Switches Under The Access Cover (left to right)

Check to be sure the printer has been set to the proper switch positions for use with a Xerox 820-II. Check your printwheel to determine if its plastic or metal and 10 or 12 pitch. The available setting positions are as follows:

1. **Printwheel Select Switch.** Set this switch to match the particular type of printwheel being used. This ensures your text will print correctly and prevents possible printwheel damage or excessive wear.

   PRINTWHEEL SELECT:

0:	88 Metal
2:	92 Metal
3:	96 Metal
4:	96D Metal
5:	APL Metal
6:	APL Plastic
7:	Plastic (This is the printwheel normally shipped with the printer)
1,8,9:	Optional

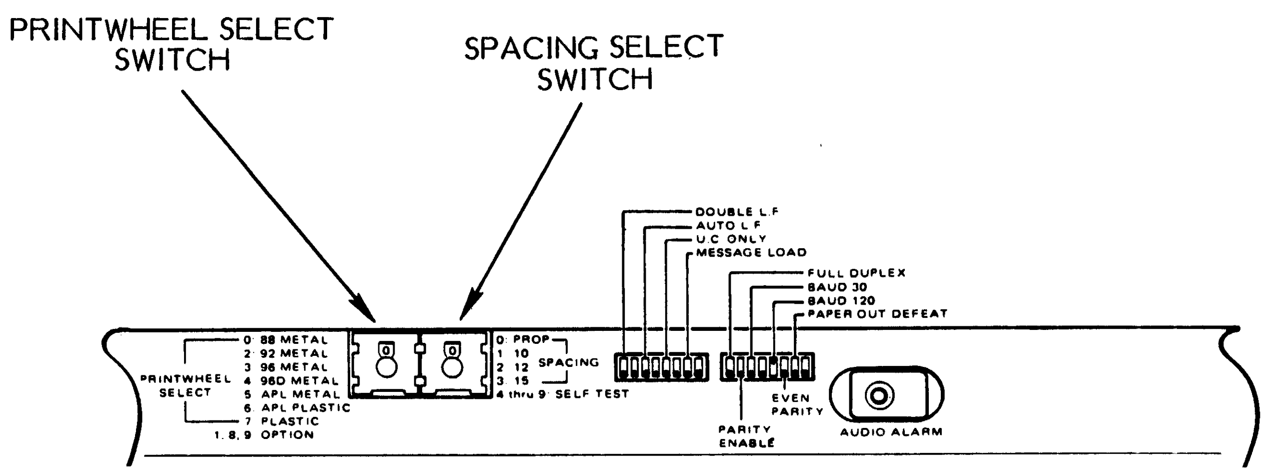
2. **Spacing Select Switch.** This switch selects the horizontal spacing for character print out. Set this switch to 1 for 10 Pitch PWS or 2 for 12 Pitch PWS.

0:	Proportional
1:	10 (This is the printwheel normally shipped with the printer)
2:	12
3:	15
4 - 9:	Self Test

3. For use with the Xerox 820-II, the switches to the right of Printwheel and Spacing switches should be positioned toward the front of the printer, except for the BAUD switch marked 120. It should be positioned toward the back of the printer.

4. Replace the access cover.

5. If your installing the printer for the first time, return to the 820-II installation card for further instructions.



Setting The Operating Switches

These six switches are located in the right-hand area of the control panel where they are accessible to the operator with all covers on the machine. These are membrance type momentary action switches actuated by a touch of the finger.

1. **RESET Switch.** This switch will restore the printer to normal operating status following a printer check or an error condition, and clears all error indicators.

2. **SCROLL Switch.** Touching this switch advances the paper a small amount to give the operator a clear view of the last printed line. The paper is automatically returned to the last printing position when the switch is released.

3. **LINE FEED Switch.** Touching this switch initiates a single or a double line feed operation, as selected by the DOUBLE L.F. MODE SWITCH. Action is repeated if the switch is held activated longer than 600 msec. A line feed code will not be transmitted.

4. **FORM FEED Switch.** Touching this switch initiates a form feed to the next top-of-form position. A form feed code is not transmitted.

5. **HERE IS Switch.** Touching this switch causes a special "Here Is . ." message of up to 31 characters to be transmitted over the communications link when operating in remote ASCII mode with the fully featured HPRO5 option installed. This is not used with the Xerox 820-II.

6. **BREAK Switch.** Touching this switch causes a break (250 msec space) to be transmitted over the communications link when operating in remote mode.

7. **Audio Alarm.** This device buzzes briefly to indicate the occurrence of various errors or operating conditions.

Reading The Control Panel Indicators (left to right)

1. **POWER.** Indicates that AC power is applied to the 630 Printer (40 CPS).

2. **PRINT CHK*.** Indicates that a print operation has been called for while the printer is in a "check" condition. A check condition occurs when a printwheel or carriage movement command has been received but cannot be successfully completed due to a malfunction. This condition disables the printer until a restore sequence clears the check condition.

   RESET. Note that if the problem causing the check condition has not been corrected when a restore sequence has been initiated, the check will reappear as soon as printing is attempted.

3. **PARITY.** Indicates detection of any of the following types of errors:

   - Incorrect parity sensed on received character.
   - A framing error (no stop bit) detected on a received non-break character.
   - A serial data character detected with an excess number of bits.

   When a parity error is detected, a DEL character is substituted for the erroneous character.

   This indicator functions only if the PARITY ENABLE switch (under the access cover) is ON.

4. **OVERFLOW*.** Indicates that the printer's print input memory (buffer) is too full (has overflowed). Protocol has not been used properly.

5. **RIBBON/PAPER*.** Indicates end of ribbon has been reached or that the printer is out of paper, and printing has been attempted.

6. **COVER*.** Indicates that printing was attempted with the sound cover open.

   * These errors cause a break to be transmitted when the 630 Printer (40 CPS) is in Remote mode if DC1/DC3 protocol has not been selected.

SPECIFICATIONS

Print Speed: Up to 40 characters per second with metalized printwheels.

Character Set: 88, 92 or 96 printable characters per printwheel. Switch selectable program support for APL and all ENGLISH language printwheels.

Printwheels:	Metal	-88 character Xerox
		-96 character Diablo and Xerox
	Plastic	-96 character Diablo

Character Spacing: 10 and 12-pitch.

Column Spacing: 1/120 inch (.21mm) minimum.

Print Line: 13.2 inches (335.3mm)
132 columns 10-pitch
158 columns 12-pitch
198 columns 15-pitch

Paper Width: 16.53 inches (419.9mm) maximum.
- friction feed without Top Paper Out switch.
16.00 inches (406.4mm) maximum.
- friction feed with Top Paper Out switch.
15.25 inches (387.4mm) maximum.
full width with optional forms tractor (14.75 inches/-374.7mm between holes). 3.25 inches (82.55mm) minimum with forms tractor (2.75 inches/69.85mm between holes).

Carriage Speed: 400msec maximum for 13.1 inches (332.77mm) of motion.

Tabulation: Left or right.

Line Spacing: 1/48 inch (.53mm) minimum.

Paper Feed: Bidirectional, except with unidirectional forms tractor and unidirectional pin feed platen.

Paper Feed Speed: 4 inches (101.6mm) per second plus 40msec (typical) settling delay time.

Sensors: End of ribbon, top paper out, and cover open.

Paper Thickness: .000 - .101 inch (.254mm) at low setting (1-3 part forms)
.010 - .027 inch (.254 - .686mm) at high setting (4-6 part form).

Other Features: Self test; host program control THROUGH escape sequences; data receive/transmit speed selection.

Power Requirements: Strappable for operation from nominal 100, 120, 220 or 240 volt (+10%/-15%) AC inputs, 49-61 Hz. 350W maximum power consumption. Factory preset for 120 VAC.

CHECK YOUR PRINTER's SERIAL PLATE FOR PROPER INPUT POWER.

This page intentionally left blank

FORMS TRACTORS

The Forms Tractor can only be installed on the 630 Printer (40 CPS).

The unidirectional and bidirectional Forms Tractors are very similar except the unidirectional does not have the reverse drive sprockets required for bidirectional paper feeding.

Unidirectional Forms Tractor: This unit mounts on top of the printer cover where it engages the platen shaft for alignment and drive. It requires use of a friction feed platen. It is adjustable to any paper width from 2-3/4" (69.85mm) to 14-1/2" (368.3mm) maximum as measured between the pin feed drive holes.

Bidirectional Forms Tractor: This unit mounts on top of the printer cover where it engages the platen shaft for alignment and drive. It requires use of a friction feed platen. It is adjustable to any paper width from 2-3/4" (69.85mm) to 14-1/2" (368.3mm) maximum as measured between the pin feed drive holes. It also features both forward and reverse pin feed paper drives to enable feeding paper in either direction.

Forms Tractor Installation

Note: If your 630 Printer (40 CPS) has a sound panel, it must be the special sound panel designed for use with a Forms Tractor.

1. If you have a sound panel, swing it forward to its open position.

2. Pull the paper release lever and paper bail toward the front of the printer.

3. Holding the forms tractor at both ends, depress the two latch release levers on the tractor and lower it onto the platen shaft . As the tractor is being lowered, guide the paper release actuator fork on the tractor over the paper release lever on the printer.

4. Release the latch levers to clamp the tractor assembly onto the platen shaft. Check to see that both ends of the tractor are firmly latched.

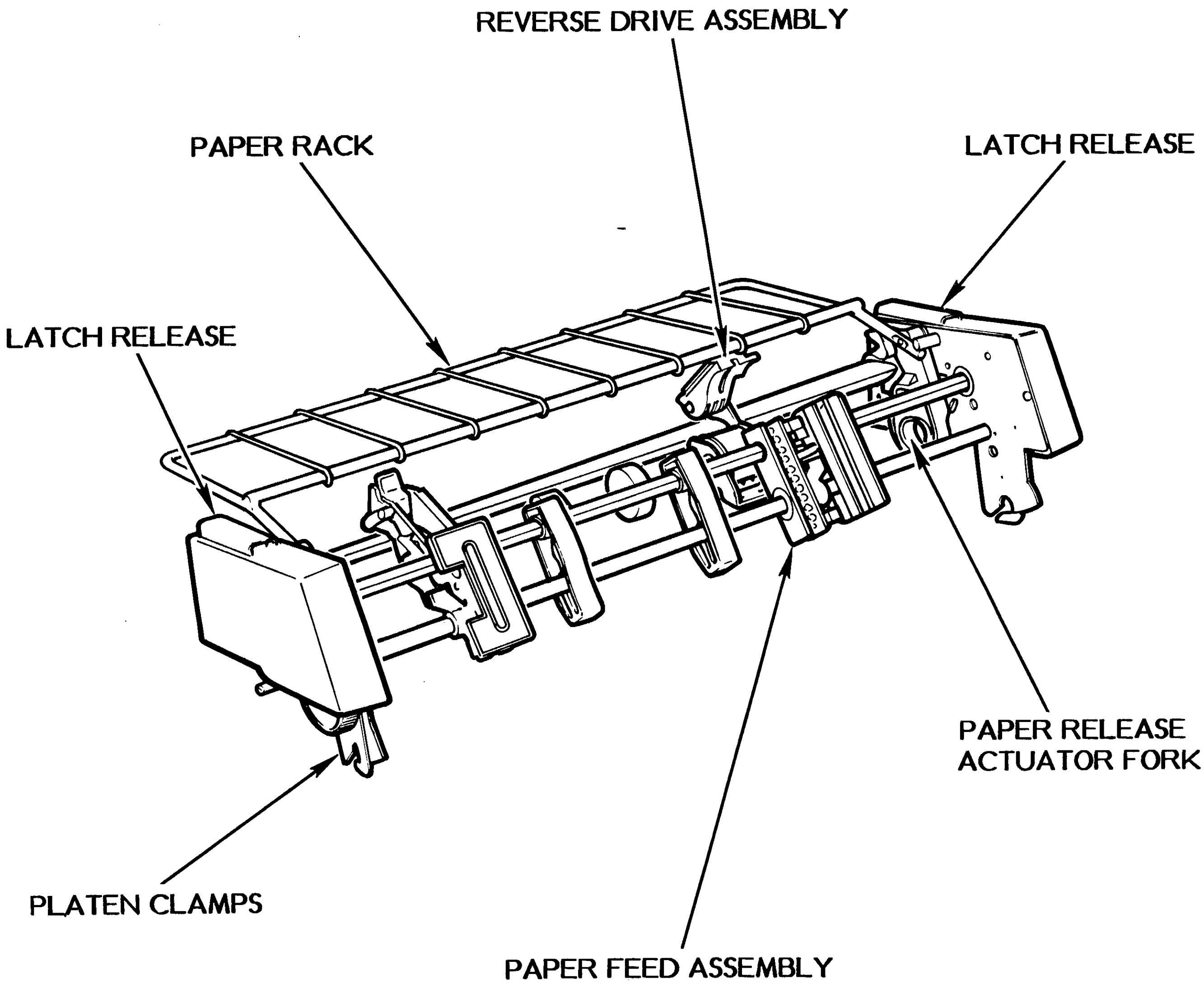
5. Swing the paper support rack forward then backward to make sure the paper release lever stays inside the release actuator fork. If the paper release lever slips out, remove the forms tractor and repeat steps 2 through 4 of this installation procedure.

Forms Tractor Removal

1. Remove any paper from the Forms Tractor.

2. Swing the sound panel forward.

3. Depress the two latch release levers and lift the tractor straight up.


REVERSE DRIVE ASSEMBLY
PAPER RACK
LATCH RELEASE
LATCH RELEASE
PAPER RELEASE
ACTUATOR FORK
PLATEN CLAMPS
PAPER FEED ASSEMBLY

Loading Paper into the Bidirectional Forms Tractor

1. If you have a bidirectional Forms Tractor, swing the sound panel open.

2. Adjust the two pin feed assemblies to the appropriate form width by loosening the feed assembly lock levers and sliding the feed assemblies to the left or right as necessary.

3. Swing the paper rack on the tractor to a forward position. This allows you to insert the paper into the reverse drive sprockets; it also moves the pressure release lever backward so the platen will be able to grip the leading edge of the paper and pull it around to the forward drive assemblies.

4. Open the gates on the reverse and forward drive assemblies.

5. Bring the leading edge of the continuous form paper up over the rear of the printer and hook the pin feed holes along each side of the paper onto the feed pins on the reverse drive sprockets. (Be certain that the paper is aligned straight on the sprockets.) Close the gates over the reverse drive sprockets to hold the paper in place on the feed pins.

6. Slowly rotate the platen to feed the leading edge of the paper down behind the paper bar, around and up in front of the platen.

7. Swing the paper rack back. This moves the paper release lever on the printer to release the grip of the platen on the paper.

8. Grasp the leading edge of the paper and pull it up while manually turning the platen until the leading edge of the paper is above the forward drive assemblies on the tractor.

9. While gently creating tension on the paper by pulling up on the leading edge, fit the side holes in the paper onto the feed pins of the forward drive belts. Close the gates to hold the paper in place on the feed pins. Be certain that the paper is aligned straight.

10. Move the paper bail back toward the platen. (The bail will be held away from the platen slightly; this is proper when operating with the forms tractor.)

11. Rotate the platen 2 or 3 turns forward and backward to check that the paper is feeding properly through the drive assemblies.

12. Close the sound panel, if you have one.

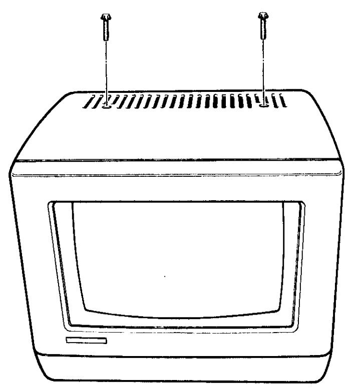
PARALLEL PRINTER INSTALLATION

Follow the instructions below to install a parallel printer on your 820-II. Before you can connect the printer, you'll need to take the cover off the 820-II screen.

TURN the 820-II off, then UNPLUG the 820-II from the wall outlet and wait 30 seconds.

WARNING: Hazardous voltage areas near the back of the screen tube will be exposed when the screen cover is removed.

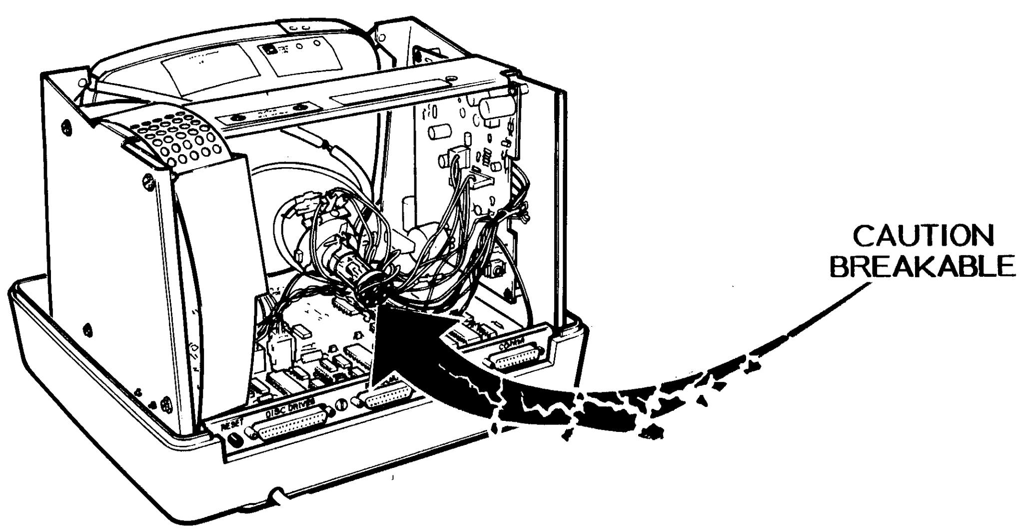
REMOVE the two screws on the top of the 820-II screen cover.



LIFT the cover straight up and place it on your table or desk.

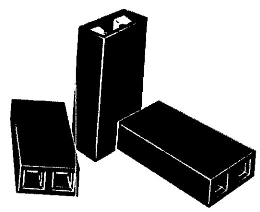
TURN the 820-II so you can see inside the back, as shown below.

NOTICE the location of the end of the screen tube. While you are working on the 820-II you must take CAUTION not to bump or damage this area. If broken, the screen tube could explode.

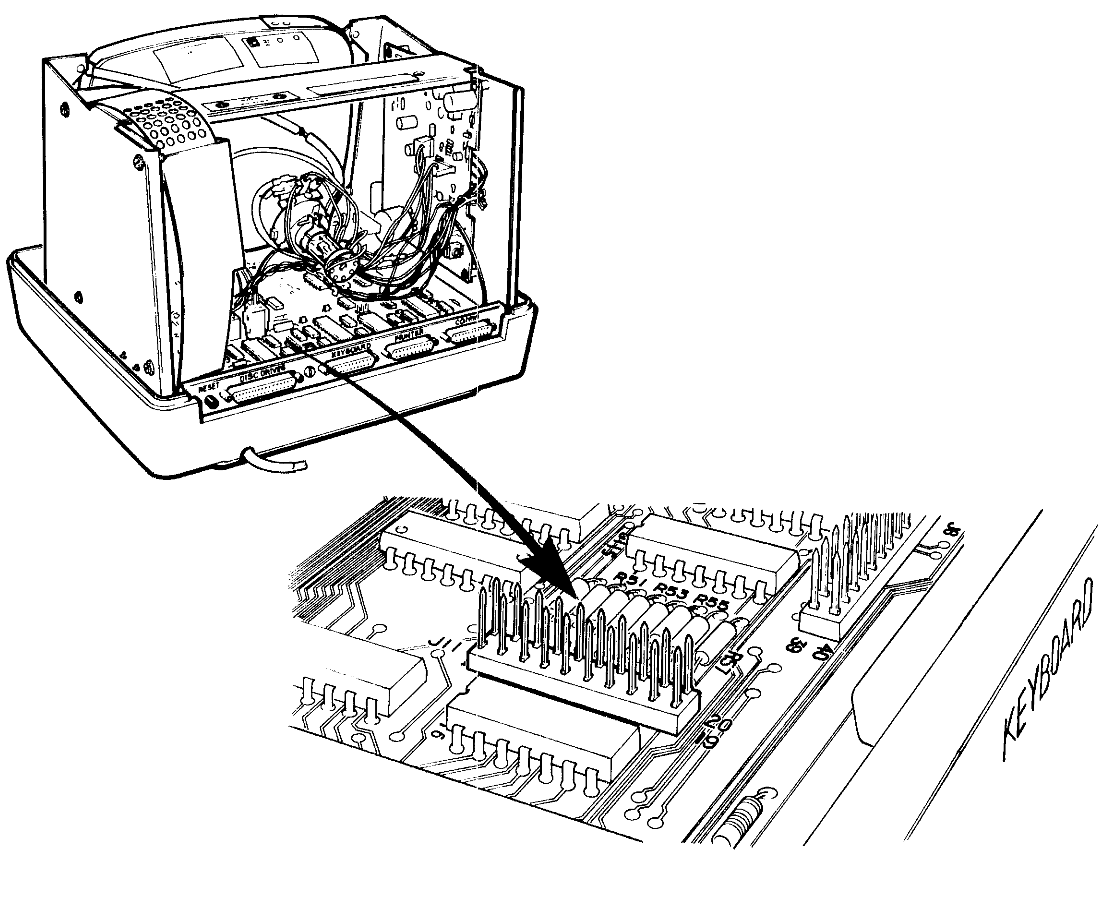


Now you're ready to install the "jumpers" (shown in the first illustration below) on the pins on the 820-II's board (as shown in the second illustration below). The pins on the board are sharp, so take **CAUTION** not to prick your fingers on them.

LOCATE the three small, black "jumpers" that came in the printer kit.

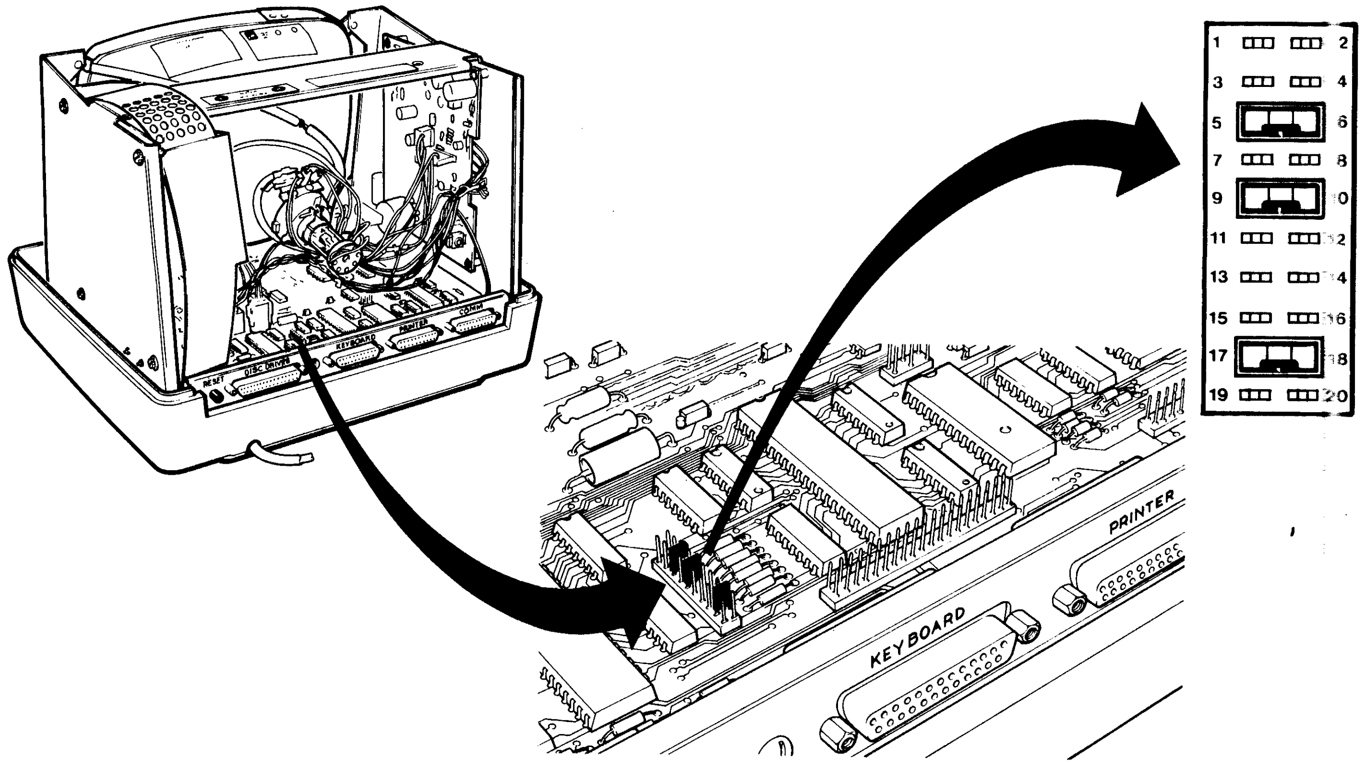


LOCATE the pins that stick up near the back of the 820-II, as shown in the illustration below.



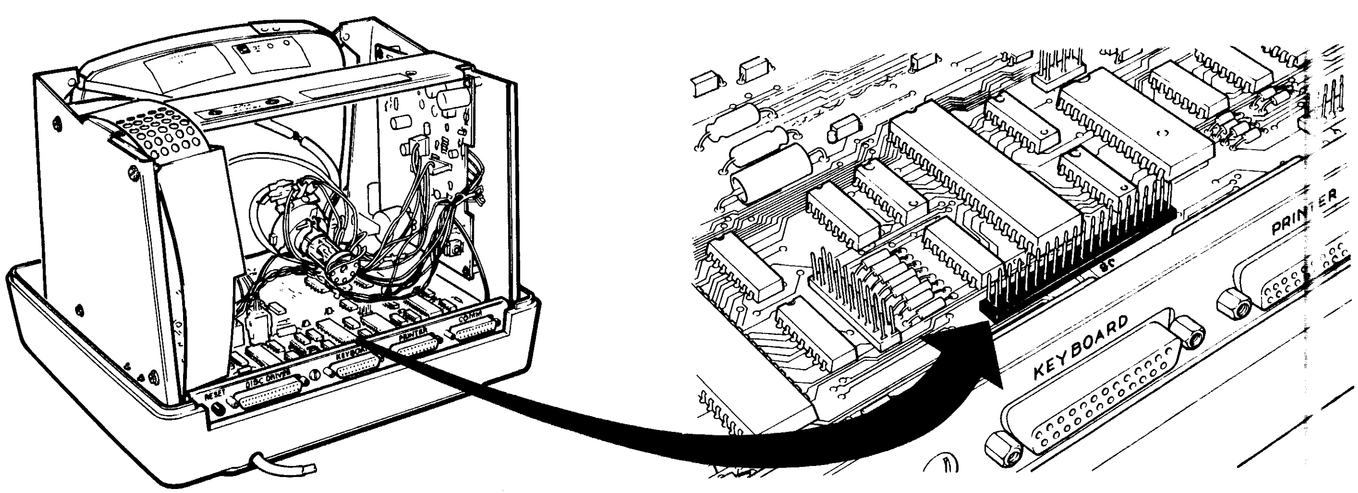
WARNING: Computer components and connectors are delicate. Care should be taken not to bend or damage these parts.

PUSH the jumpers onto the third, fifth and ninth rows of pins, as shown below. The illustration (below right) shows that the jumpers should cover pin numbers 5 and 6, pin numbers 9 and 10, and pin numbers 17 and 18.



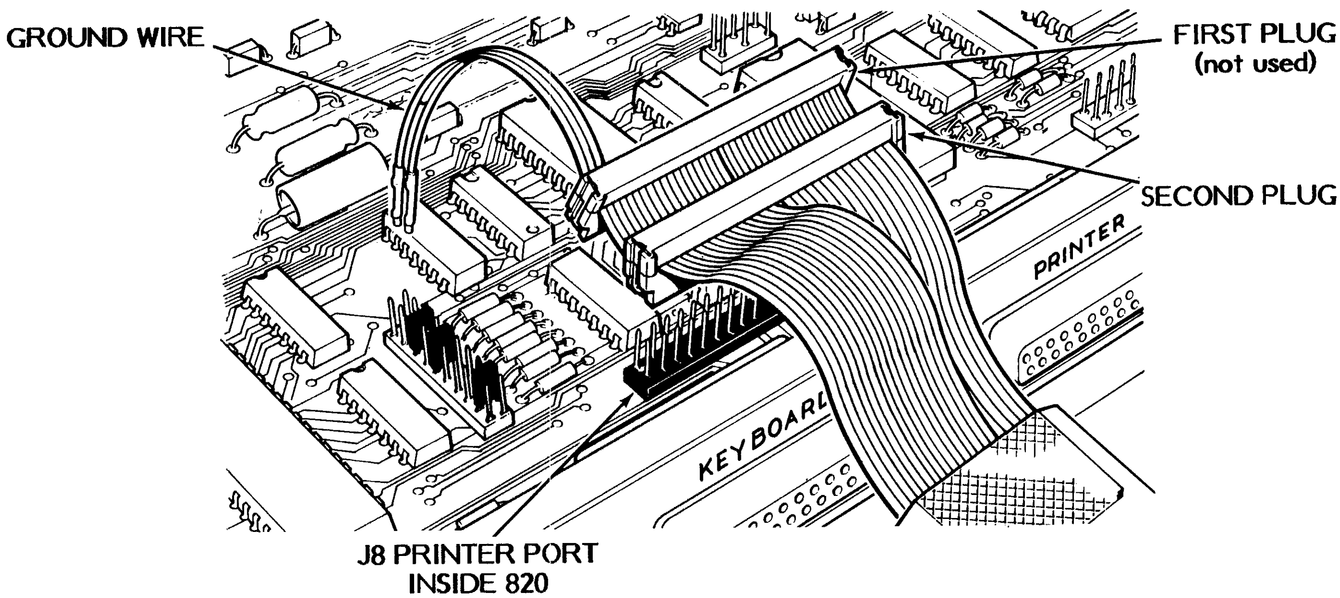
The parallel printer is connected to a printer port inside the 810. It is not connected to the port on the back of the 820-II.

LOCATE the parallel printer port (marked J8) inside the 820-II, as shown below.



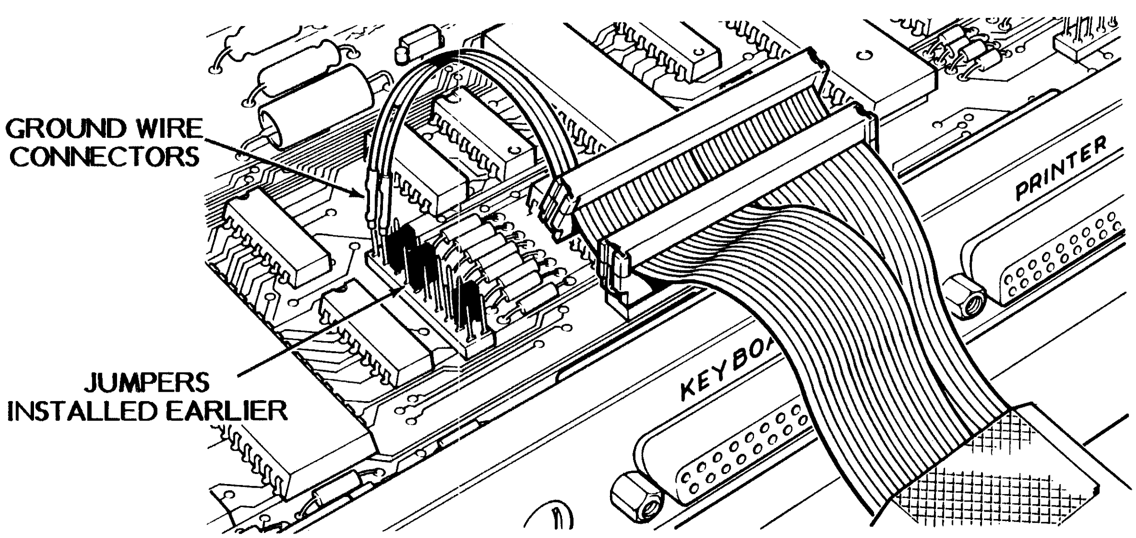
LOCATE the cable that came with your printer kit. Notice that one end of the cable has two small plugs, with a ground wire coming off the first plug.

PUSH the second small plug on the cable down on the printer port inside the 820-II, as shown below. Be sure that the plug covers both rows of pins, then push the plug down firmly.

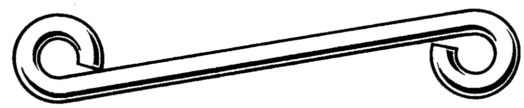


LOCATE the ground wire.

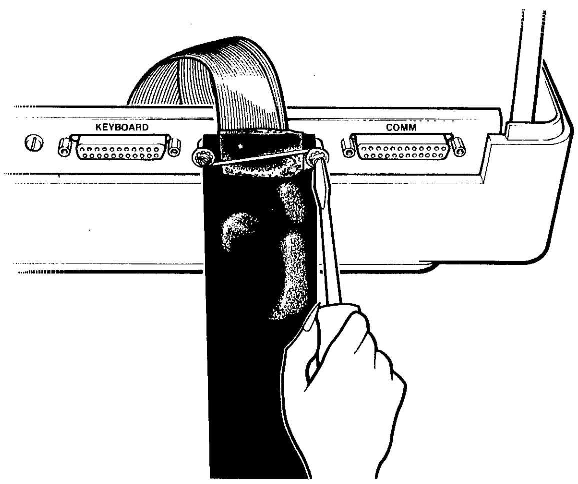
PUSH the two ground wire connectors onto the pins in front of and to the <u>left</u> of the first jumper you installed earlier. Be sure the ground wires connect to the exact pins shown in the illustration below.



LOCATE the wire bracket (shown below) that came with your printer kit.



PLACE the bracket over the copper foil on the printer cable and screw the bracket into the holes beside the socket marked "PRINTER" on the back of the 820-II. (See the illustration below.)



Now that you've installed your printer, you may want to go back and double check your work before you put the cover on the 820-II.

REPLACE the cover on the 820-II and fasten it in place using the two screws removed earlier from the top of the cover.

PLUG the other end of the printer cable into your parallel printer.

PLUG the 820-II back into a wall outlet.

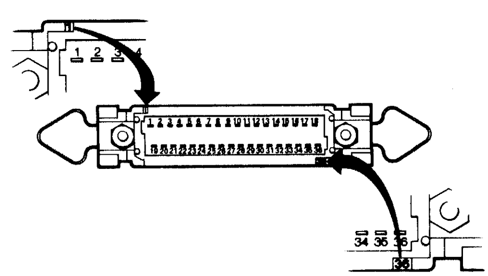
CHECK your printer for a power cable. You'll need to plug it into a wall outlet and turn your printer on before you print.

NOTE: It is a good idea to check your printer manual for the printer set-up information (such as the DIP switch settings), if you have not already done so. Suggested settings for some printers are given on page 16.

XEROX PARALLEL CABLE

PRINTER CONNECTOR	PRINTER SIGNAL NAME	SOURCE	820 SIGNAL NAME	820 CONNECTOR	
P1				J8	
1	Data Strobe	820	PB2	30	
2	Data Bit 1	820	PAØ	6	
3	Data Bit 2	820	PA1	8	
4	Data Bit 3	820	PA2	10	
5	Data Bit 4	820	PA3	12	
6	Data Bit 5	820	PA4	14	
7	Data Bit 6	820	PA5	16	
8	Data Bit 7	820	PA6	18	
9	Data Bit 8	820	PA7	20	
10	Acknowledge	Printer	PB7	40	
11	Busy	Printer	PB4	34	
12	NC		PB5	36	
13	On Line	Printer	PB6	38	
14	Auto LF	820	PBØ	26	
15	NC				
16	GND	820		37	
17	NC				
18	NC				
19	GND	820		1	Cable
20	GND	820		5	
21	GND	820		7	
22	GND	820		9	
23	GND	820		11	
24	GND	820		13	
25	GND	820		15	
26	GND	820		17	
27	GND	820		19	
28	GND	820		21	Cable
29	GND	820		3	
30	GND	820		35	
31	NC				
32	NC				
33	GND			39	
34	NC				
35	NC				
36	NC			28	

Below is a top wire view of the 36 pin Centronics connector.



DAISY CHAINING DISK DRIVES
(Connecting)

Use this procedure for connecting 8" floppy disk drives and a rigid disk drive to an 820-II.

CAUTION: If you are adding a rigid disk drive to an 820-II that was originally installed with floppy disk drives, make sure that your screen has been upgraded to operate with a rigid disk drive **before** connecting it to the back of the screen. If a rigid disk drive is connected to a screen that is not upgraded, it will cause permanent damage to the screen.

If you've just received the rigid disk drive, check to be sure that the shipping (locking) bracket has been removed from the rigid disk drive spindle.

Connecting the Disk Drives

TURN the 820-II off, then UNPLUG the screen from the wall or floppy disk drives.

IF you are upgrading your system, disconnect the floppy disk drive cable from the back of the screen and remove the power cord from the wall.

CONNECT the rigid disk drive cable to the outlet marked "DISK DRIVES" on the left side of the back of the screen. Use a screwdriver to tighten the screws on both sides of the connector. (Be careful not to over-tighten the screws).

PLUG the power cord into the back of the rigid disk drive, then into a power outlet.

LOCATE the large cable connector on the floppy disk drives and connect it to the back of the rigid disk drives and tighten the screws on both sides of the connector.

CONNECT the power cord into the back of the floppy disk drives, then plug it into the accesory outlet on the back of the rigid disk drives.

PLUG the printer into a power outlet or into the accesory outlet on the back of the floppy disk drives if a wall outlet is not available.

Follow the directions in the DIAGNOSTIC section of your 820-II Reference Guide to be sure your 820-II is in proper working order.

SYSTEM RESIDENT MONITOR

The XEROX 820-II Basic Operating System monitor (BOS) is the primary control program for the Xerox-II. It begins execution when the computer is first turned on, or whenever the reset button is pressed. The monitor resides in the top 4K of read-write memory, and also in the lower 6K of Read Only Memory (ROM).

BOS provides several essential functions for the system. It is the initial software level of the computer and contains routines to control the initialization operation of the system input/output and memory resources. The initial functions of BOS include commands to display and alter the contents of memory and I/O ports, to begin execution at a given address with user-specified entry conditions, and to load programs from any one of the disks. The I/O functions of BOS provide driving routines for the built-in CRT display and keyboard, the printer and communications ports, and the disk controller. In this capacity BOS is always active, even when applications programs like the CP/M disk operating system have control of the CPU.

The following sections of this manual will explain how to use the BOS Console Monitor commands, what facilities are provided by the resident I/O handlers, and how to interface applications programs with the BOS.

XEROX 820-II COMMAND SUMMARY

The Xerox 820-II BOS enters the BOS Console Monitor mode after it has initialized the system following a power-on or a reset. The following sign-on message is displayed on the screen as an indication that BOS is ready to accept commands.

```
820-II v 0.00 (C) 1982 Xerox Corp

L - Load System
H - Host Terminal
T - Typewriter

*
```

Commands consist of a single character command name and parameters. The parameters are hexadecimal numbers and can be separated by commas or spaces. The command line may be entered using upper case or lower case letters. The RETURN key terminates the command entry. Errors within a line can be corrected by touching the BACK SPACE key to delete the last character or CTRL + X to delete the entire line. If an invalid command or parameter is detected, an error message will be displayed and the command will not be executed.

The user may wish to halt long running commands like the memory display before they are finished. This can be done by touching the RETURN key while the command is displaying on the screen. The display can also be stopped temporarily by touching the space bar. Touching the space bar again will allow the display to continue.

The following table summarizes the monitor's command set. The items enclosed in angle brackets represent the numeric parameters required by the command. The items enclosed in square brackets represent optional parameters.

Command	**Format**
d(isplay memory)	D start address end address
m(odify memory)	M address
x(tended memory test)	X start address end address
f(ill memory)	F start address end address fill data
c(opy memory)	C start addr end addr destination addr
v(erify memory block)	V start addr end addr with addr
g(oto)	G address HL DE BC registers
i(nput)	I 16-bit port address
o(utput)	O 16-bit port address data
l(oad from disk)	L disk drive unit value
r(ead disk sector)	R drive unit track sector address
b(aud rate)	B baud rate code channel
t(ypewriter)	T baud rate code
h(ost terminal)	H channel
p(rinter protocol)	P Xon/Xoff status mask status value

1) D - DISPLAY MEMORY COMMAND

The display memory command will display of the contents of memory in hexadecimal and ASCII representation. Each display line has the following format:

AAAA DD DD DD DD DD DD DD DD DD DD DD DD DD DD DD DD CCCCCCCCCCCCCCCC

where AAAA is the starting memory address of the line in hexadecimal, the DD's are the hex values of the 16 bytes of data starting at location AAAA, and the C's are the ASCII characters equivalent to each data byte. Bytes with a value less than 20 hex are displayed with their appropriate display character code as shown in the display character code table. Bytes with a value greater than 7F hex are displayed in low intensity. The display memory command accepts zero, one or two address parameters. If two addresses are specified, the block of memory between those two locations will be displayed. Entering only one address will display 256 bytes of memory starting at the specified location. Entering 'D' with no parameters will display the 256 byte block of memory starting one location past the last address displayed by the previous display memory command.

2) M - MODIFY MEMORY COMMAND

The modify memory command allows the contents of individual memory locations to be changed. This command accepts one parameter representing the first memory address to modify or examine. The display format is:

AAAA DD

where AAAA is the current memory address and DD is the hexadecimal value of the data in that location. After displaying the contents of a memory location, the routine waits for one of the following parameters to be input from the keyboard:

- Touching the RETURN key will not modify the memory data at the currently displayed memory address, but will display the contents of the next memory address.
- Typing a minus sign will have a similar effect, except the address is decremented instead of incremented.
- Typing a hexadecimal number will replace the data at the currently displayed address with the number entered. The new data is stored as soon as the second digit is entered, with no terminating character required. If only one digit is entered, touching the RETURN key will cause the single digit hex number to replace the previous data.
- Typing a quote sign will cause the ASCII value of the next key typed to be stored at the currently displayed address.
- Typing any character other than a RETURN , a minus sign, a quote sign, or a hexadecimal digit will terminate the command.

3) X - EXTENDED MEMORY TEST COMMAND

This command will test the specified range of memory for errors that may be caused by any of a number of reasons. Any portion of memory may be tested except the read-write area reserved for BOS (F000 to FFFF hex). At least two parameters are required, the starting address and ending address to be tested.

Only the high order eight bits of the addresses entered are actually used, however. If no errors are detected, the test will display a plus sign every time a test pass is completed. A total of 256 plus signs must be displayed for all possible test patterns to have been tried. When errors are detected an error line will be displayed in the following format:

AAAA DD should=XX

where AAAA is the address of a location that fails the test, DD is the data read back from the defective location, and XX is the test pattern that was written there.

4) **F - FILL MEMORY COMMAND**

The fill command allows blocks of memory to be filled with a fixed data value. Three parameters are required in the command line; a starting memory address, an ending address, and a fill data value. Each location in the specified block of memory has the constant written into it and then read back again to check for memory errors. An error line like the one described for the 'X' command is printed for any locations that fail to verify.

5) **C - COPY MEMORY COMMAND**

The copy command allows blocks of data to be moved in memory. Three parameters are required in the command line; a starting memory address, an ending address, and a destination address. The contents of the block of memory bounded by the first two addresses is copied to the block starting at the third address. As with the fill command, a test is made to verify that each byte of the destination block, when read back, is the same as the corresponding byte in the source block.

6) **V - VERIFY MEMORY BLOCK COMMAND**

This command is similar to the copy memory command except that memory data is not moved, but simply checked to see if it is the same as data that is located at a different address in memory. Three parameters are required; a starting memory address, an ending address, and the first address of the memory block to compare with.

7) **G - GO TO COMMAND**

The go to command allows control of the CPU to start executing at a particular memory location. This command requires a single parameter from the user representing the address at which to begin execution. Three additional optional parameters may also be specified so that the HL, DE, and BC register pairs may be preset before execution begins at the specified address. Each of these optional parameters is a 16-bit (four digit hexadecimal) number. A command line using the optional parameters would be entered as:

G1000 AAFF DDEE BBCC

where 1000 is the hexadecimal address at which to begin execution, AA is placed in the H register, FF is placed in the L register, DD is placed in the D register, EE is loaded into the E register, BB is transferred to the B register, and CC is put into the C register. In addition, FF (the value specified for the L register) will also be placed into the A register. Thus, a shortened command line using only a single optional parameter would be entered as:

GF00C 0024

In this example, the hex value 24 will be loaded into both the L and A registers before executing the CRT output driver at memory address F00C hex. This example will display a '$' on the screen.

BOS actually passes control to the specified address by simulating a CALL instruction. This makes it possible for the external program to return to BOS by doing a RET, assuming it does not re-load the stack pointer or lose the return address to BOS. After the routine returns, BOS will display the contents of the A register, and the HL register pair.

8) I - INPUT COMMAND

This command allows data to be read from input ports. It operates very much like the modify memory command, except that input ports are being examined instead of memory locations. A single parameter representing a port number is expected in the command line. Since many of the I/O ports of the XEROX 820-II are accessed using the unique Z-80 I/O instructions, the parameter may be a 16-bit port address. The BC register pair is loaded with the parameter, and then an IN A,(C) instruction is executed. An example of the full parameter specification would be:

IAA55

where AA represents the contents of the B register, which is placed on the high order address lines (A8-A15), and 55 represents the contents of the C register, which is placed on the low order address lines (A0-A7). Using only an 8-bit parameter will place a zero in the B register.

Touching the space bar will input data from the same port address again. The contents of adjacent ports can be examined by touching the RETURN key or a minus sign as in the 'M' command. Typing any other key will terminate the command.

9) O - OUTPUT COMMAND

The output command allows output ports to be written with a specified data value. Two parameters are required in the command line; a 16-bit port address as described for the input command, and an 8-bit data value that is to be written to that port. After outputting the specified data to the port, the command returns to BOS instead of stepping to the next output port like the input command. This makes it possible to use the output command to initialize Z-80 peripheral devices like the SIO, PIO and CTC. Since a 16-bit port address is specified, special ports such as the scroll port register can be modified directly from BOS. Some of these special ports require that their "data" value be placed on the high order address lines. A sample command to alter the scroll port register is:

O1014 FF

where 10 represents the contents of the B register which is placed on the high order address lines and is the actual "data" that will be written to the scroll port register. The 14 represents the value that is placed into the C register and is output as the low order address lines to actually select the scroll port. The data value FF hex is output on the data lines, but the data is not looked at by this type of special port.

10) L - LOAD SYSTEM

The load from disk command is used to read a one-sector program from the first sector of the specified disk drive. The most common use of this command will be to load the CP/M disk operating system, though it is not solely restricted to this purpose. The load command accepts one optional parameter to specify which physical disk to load from. If this optional parameter is omitted, BOS will load from physical drive unit 0, or the "A" disk drive. The parameter may be in the range of "A" through "P", which represents the valid disk drive designations for CP/M.

Configurations with only floppy disks have valid disk drive parameters of "A" through "D". The normal load from disk command for floppy drives will simply be L or L A, to load from drive "A".

Configurations with a rigid disk drive can have valid disk drive parameters of "A" through "H".

One of the most powerful features of BOS is that the drive that is loaded from becomes logical drive "A:" in CP/M. Thus when CP/M is loaded from drive "E" (the L E command), the disk will be referenced in CP/M as A: so that the high speed of the rigid disk is available for warm starts. See the notes on Xerox CP/M for a more detailed explanation of logical drive swapping.

The disk loader reads the first logical sector into memory at location 80 hex and starts execution at that address. Normally the program will be a small loader that in turn reads in a larger program such as CP/M. This two level bootstrap process makes the boot command more application independent. The only requirements are that the first sector of the disk be reserved for a loader, the first instruction of this loader cannot be a PUSH HL, and the first 256 bytes of memory cannot be overwritten by the program being loaded.

11) R - READ DISK SECTOR COMMAND

The read disk sector command allows one physical sector to be read from the specified disk drive to a designated address in memory. The drive unit is a number between 0 and F hex, with 0 corresponding to the "A" drive as described in the load from disk command.

It should be noted that different disks may not have the same sector size. The read sector command will always read one physical sector, no matter what its length. Typically single density disks will have 128 byte sectors, and double density disks will have 256 byte sectors. The sector size for rigid disk drives will always be 256 bytes or larger. Even though one physical sector is read, 256 bytes will be displayed after the read. Thus, when reading single density disks, only the first 128 bytes of the 256 bytes displayed on the screen are valid.

12) B - BAUD RATE COMMAND

This command will set the baud rate for the designated serial I/O channel. One parameter is required to indicate the baud rate code as shown in the following table. If the second optional parameter is not specified, then the baud rate for the printer port (channel B) will be set. Specifying the "A" channel will cause the baud rate for the communication port to be set. At power-on or reset, the communications port is set for 300 Baud, and the printer port for 1200 Baud. Both serial ports are set for 7 data bits and even parity with one stop bit.

Baud rate code (hex)	Baud rate
00	50 Baud
01	75 Baud
02	110 Baud
03	134.5 Baud
04	150 baud
05	300 Baud
06	600 Baud
07	1200 Baud
08	1800 Baud
09	2000 Baud
0A	2400 Baud
0B	3600 Baud
0C	4800 Baud
0D	7200 Baud
0E	9600 Baud
0F	19.2 Kbaud

13) T - TYPEWRITER COMMAND

This command allows the XEROX 820-II to be used as a standard electronic typewriter. All keystrokes will be typed directly on the attached printer and the display screen in a direct print mode. The optional baud rate code parameter can be used to alter the printer channel baud rate so that two commands are not necessary to use printers with Baud rates that are different than the default rate of 1200 baud. To exit the typewriter mode, touch the CTRL+ESC keys.

14) H - HOST TERMINAL COMMAND

The host terminal command configures the XEROX 820-II as a sophisticated editing terminal when connected to another Host Computer or time share system. Two optional parameters may be specified when entering the terminal mode to indicate which serial channel to use. If no parameter is specified, the communication port will be used. To use the printer port, use "B" as the optional channel parameter (a step by step procedure for using the Host Terminal Mode is provided in the General Programs section of this manual).

A unique feature of the XEROX 820-II is that all information displayed is saved in unused memory as the information is scrolled off the top of the screen. Four command keys control how this stored information can be recalled:

- CTRL+DOWN ARROW will cause information scrolled off the top of the screen to reappear line-by-line at the top of the display. Information that scrolls off the bottom will be saved.
- CTRL+UP ARROW will cause information that has scrolled off the bottom of the screen to be recalled line-by-line at the bottom of the display.

The terminal operates initially in full duplex mode. Characters typed on the keyboard will not be displayed on the screen unless the Host Computer has received the typed character and transmitted it back to the XEROX 820-II. However, the 820-II can be operated in several different modes which are activated by using the CTRL key in conjunction with the keys on the **numeric keypad:**

- CTRL+DEL Enable local echo. Characters typed on the keyboard will be displayed on the screen and transmitted through the serial port. Touching these keys again will disable the local echo mode of operation.
- CTRL+LINE FEED Enable local auto line feed. When the RETURN key is touched, a line feed will also be sent to the local screen display, but not transmitted through the serial port. Touching these keys again will disable the local auto line feed mode of operation.
- CTRL+1 Enable remote echo. Characters received through the serial port will be echoed back to the transmitting device. In this mode the XEROX 820-II may act as a host to another terminal. Touching these keys again will disable the remote echo mode of operation.
- CTRL+2 Enable remote auto line feed. Carriage return characters received through the serial port will be echoed to the remote device as carriage return and line feed codes. Touching these keys again will disable the remote auto line feed mode of operation.
- CTRL+. Transmit BREAK. When the CTRL and the period key on the numeric keypad are touched, a break condition will be enabled on the serial port until: 1) CTRL+. is touched again. 2) Any other character is typed. "Toggling" the break function in this way allows the length of the break condition to be determined by the user. Some host computers require a very short break condition, while some communication control devices require a long break condition.

In the terminal mode the Xerox 820-II will respond to the special control codes and escape sequences described on page 16. These codes are similar to ADM-3A style terminals, with several enhancements for sophisticated editing control. To exit the Host Terminal mode, touch the CTRL+ESC keys.

15) **P - PROTOCOL COMMAND**

The protocol command allows the method used to control the transmission of characters to the printer to be altered for different types of serial printers. Normally the Xon/Xoff protocol is enabled to allow efficient communications with a Xerox 620 (20 CPS) or 630 (40 CPS) printer. Since this is a "transparent" protocol, it will not interfere with printers that do not use this method.

The protocol command requires at least one parameter to enable or disable the Xon/Xoff protocol. P1 enables this protocol, while P0 disables it.

The second type of protocol is used for printers that control the transmission of characters by means of a "reverse channel" or other hardware communications signals. Two signals may be used in the Xerox 820-II to control the transmission of characters to the printer:

RTS (Request to Send)	Printer connector Pin 4
DTR (Data Terminal Ready)	Printer connector Pin 20

Two parameters are used to specify how these signals will be used for "hardware handshaking"; the first designates which signals are to be checked, and the second indicates which logical state will be used to enable the transmission of data.

The most commonly used modes are shown below. The voltage level is the EIA RS-232 level measured at the printer connector:

P1 28 28	Check RTS and DTR, pins 4 and 20. If either changes to false (-12), stop transmission.
P1 8 8	Check DTR, pin 20. If false (-12), stop transmission.
P1 20 20	Check RTS, pin 4. If false (-12), stop transmission.

(The following examples show the values for some less common printers that indicate transmission should stop with signals of the opposite sense.)

P1 28 0	Check RTS and DTR, pins 4 and 20. If either changes to true (+12), stop transmission.
P1 8 0	Check DTR, pin 20. If true (+12), stop transmission.
P1 20 0	Check RTS, pin 4. If true (+12), stop transmission.

Note that all the above examples also enable the Xon/Xoff protocol by specifying a "1" as the first parameter.

USER ACCESSIBLE MONITOR ROUTINES AND VARIABLES

This section gives the locations and calling sequences of the user accessible I/O routines in the Xerox 820-II Basic Operating System (BOS). It also describes a number of important monitor variables that may need to be accessed by user written programs.

The BOS subroutines are accessed via a table of JUMP instructions beginning at memory location F000 hex. All BOS calls should be made to these entry points, since the actual addresses of the routines will vary between different releases.

Storage for the monitor's stack and working variable occupies the top 256 bytes of memory, from FF00 to FFFF hex. Programs should not attempt to write into any locations in this block that are not specifically mentioned below.

XEROX 820-II SUBROUTINE ENTRY POINTS

Most of the entry points are downward compatible with the original Xerox 820. Differences are noted with an *. Entries with a # are provided exclusively for 820-II BIOS compatibility.

*	F000	Cold Start. The resident monitor is loaded from ROM.
*	F003	Warm Start. Called by keyboard interrupt service routine when CRTL + ESC is touched.
	F006	Keyboard Status. Returns FF if keyboard ready, 00 if not.
	F009	Keyboard Input. Waits for, then returns keyboard data in A.
	F00C	CRT output. From reg A, saves/restores all registers
*	F00F	Fast CRT output. From reg C destroys registers.
	F012	SIO-B input ready status. Returns status in A register.
	F015	SIO-B input data. Returns character in A register.
	F018	SIO-B output data. Transmit character from reg A.
#	F01B	Select logical disk. From reg C (00H-0Fh).
#	F01E	Home disk. Seek track 0.
#	F021	Seek Logical Track. Track in reg C, always returns success.
#	F024	Read Sector. Transfer address in regs HL, Sector in reg C.
#	F027	Write Sector. Transfer address in regs HL, Sector in reg C.
	F02A	Execute Physical Driver. Parameter block pointer in regs HL.
	F02D	Set direct CRT cursor. Address from regs HL.
	F030	Direct CRT display. Store character in A at location set by direct CRT cursor, increments direct cursor address.
	F033	CRT memory block move. Move memory block to or from screen. Source address in HL, destination in DE, number of bytes to move in BC. Register A indicates type of move.
	F036	Return disk map table address in HL.
	F039	Return address of time of day locations in HL.
	F03C	Return configuration status.
	F03F	SIO-B output ready status. Returns FF if ready, 00 if not.
	F042	Get or Set configurable data.
	F045	Start screen print. Initiates background screen print.
	F048	User accessible 1 second interrupt.

1) 0F000 - Cold Start.
   This entry point may be called at any time to cause a Software Reset. The BOS is reloaded from ROM and all I/O devices are reinitialized.

2) 0F003 - Warm Start.
   This is actually an exit point from the BOS. It is called by the keyboard interrupt service routine when CRTL + ESC is touched. When the BOS is initialized, this exit point is set to the address of the scan line scanner. Thus, CTRL + ESC is used to exit the various command processors. When the L(oad) command enters the boot loader, it directs the exit point to the Cold Start entry point. This causes CTRL + ESC to act the same as pressing the reset button. The user may load the address of his own software abort routine into locations 0F004-0F005. This routine must be located in the upper 16K of RAM (above 0C000). Only the HL, BC, and AF registers are available for use if the routine RETurns to the keyboard interrupt driver. Any other registers used must be saved. Five levels of stack space are available. All rules of Interrupt Service Routines must be followed. For example, no calls may be made to the BOS I/O drivers. Typically this routine will set an Abort Flag that is monitored by the application and exit with a RET instruction. When the application sees the flag set, it should proceed with its abort sequence. If the behavior of this key is undesirable, simply patch a RET instruction (0C9h) at location 0F003. But, save the opcode that was there first so that it can be restored when you are finished.

3) 0F006 - Console Status.
   Returns 0FFh in the A register if console data is available. Otherwise a 0 is returned. The flags reflect the state of A.

4) 0F009 - Console Input.

5) 0F00C - Console Output.
   The character in register A is sent to the CRT driver. All registers are preserved. This entry is convenient but more time consuming than the next one.

6) 0F00F - Fast Console Output.
   This entry takes the character in register C and sends it to the CRT driver. All 8080 compatible registers are destroyed, while the additional Z80 registers are preserved. Valuable information will be returned in registers HL and A. Normally, the character under the cursor prior to entry is returned in A and registers HL contain the address in CRT RAM of the cursor in exit. However, several display control functions will return with values worth their weight in gold. A few gold mines are listed below:

   a) Character Insert. (ESC Q)
      The returned value is the character that got pushed off the end of the line.

   b) Character Delete. (ESC W)
      The returned value is the character that got deleted. These two gold mines are useful for horizontal scrolling.

c) Line Insert. (ESC E)
The line that was pushed off of the bottom is moved to the Command Processor's line input buffer. The location of this line buffer is directly after the time of day clock variables, whose address is obtained through another BOS entry.

d) Line Delete. (ESC R)
The line that was deleted was first moved to the line buffer, as in Line Insert. These golden lines are used by the H(ost) terminal command to remember lines normally lost.

e) Line Feed. (CTRL + J)
The A register returns a flag indicating whether or not the line feed caused the top line to be lost (screen scrolled). If so, the line may be found in the BOS line buffer, as in line delete.

7) 0F012 - SIO B Input Ready Status.

8) 0F015 - SIO B Input Character.

9) 0F018 - SIO B Transmit Character. (See 15, 0F0xx below)

10) 0F01B - Select Logical Disk from Register C.
The value in register C is used to select the logical disk driver in all successive 820-II emulator read or write requests. The first time this entry is called, the physical disk driver will be asked to identify the media in the selected drive. The disk must be single density since you are running an old 820 CP/M compatibility. Their behavior emulates, but does not exactly duplicate the operation of the original entry points.

11) 0F01E - Home Disk Heads.
This entry point calls the next one with a value of zero in reg C.

12) 0F021 - Seek to Track in Register C.
The value in register C is saved for later use when a read or write call is made. Note that no disk action is taken at this time.

13) 0F024 - Read Sector.
The sector specified by register C, on the track specified in the last Seek call is read into the buffer pointed to by registers HL. Note, if a double density disk is read, the entire 256 byte sector will be read in, which will explode the old 820 BIOS.

14) 0F027 - Write Sector.
The arguments are identical to the Read entry point; the sector is written from the buffer.

The previous entry points are similar to the ones provided by the original XEROX 820-II. On the next page, several new entry points for the Xerox 820-II will be described. These new functions directly control the operation of the BOS monitor. Due to the extreme power of these functions, their use should not be taken lightly. Inexperienced programmers should get some experience first.

15) F02A - Execute Physical Driver.
This entry point is the heart of the disk sub-system. All disk I/O must be dispatched by this point. Registers HL must point to a nine byte block of memory as follows:

```
00: db   command     ;0ffh = Select, 00h = Write, 01h = Read
01  ds   1           ;This byte filled in by the BOS
02  db   Ldrive      ;Logical drive for request
03  dw   Track       ;Track number for request
05  dw   Sector      ;Sector number for request
07  dw   Address     ;Address of sector buffer for request
```

The byte holding the logical drive (HL+02) is used to select the appropriate physical disk driver by indexing into the Select Table to obtain the driver unit, as well as the driver entry point address. Byte (HL+01) is filled with the physical unit number for this physical driver, then the command is passed directly to the physical disk driver. Any (pseudo) disk driver may be linked into the Select Table if it conforms to this virtual interface. The following command must be supported by the physical driver.

0FF - Select Media Format. This command causes the disk driver to identify the media in the logical drive. Registers HL return pointing to a CP/M compatible Disk Parameter Header if the media was successfully identified. Otherwise, a ZERO is returned. Since this command causes several disk accesses, it should not be issued repeatedly. The XEROX CP/M issues this command whenever a disk is "Logged In".

000h - Write Sector. This command causes the physical sector identified by bytes 03-06 of the command to be written from the buffer pointed to by bytes 07-08. The acceptable values for Track and Sector vary with different physical disk drivers.

001h - Read Sector. This command causes the physical sector identified by bytes 03-06 of the command to be written from the buffer pointed to by bytes 07-08. The acceptable values for Track and Sector vary with different physical disk drivers.

16) F02D - Set Direct CRT Cursor.
This entry stores the address passed in registers HL for use in successive calls to the next entry point.

17) F030 - Direct CRT Display.
Store character in A in the CRT RAM at the direct cursor location. The normal cursor is uneffected. The direct cursor address is incremented, however, line/screen overflow is not processed.

18) F033 - CRT Memory Block Move.
Moves a memory block to or from the alternate memory bank. If data is transferred to or from the screen RAM, only 80 bytes should be moved at a time. This is because each 80 character line actually occupies 128 bytes of address space. This entry may also be used to access the ROMs or RAM in the alternate memory bank. Length restrictions do not apply in this instance. Parameters are as for the Z80 LDIR instruction, source address in HL, destination in DE, number of bytes to move in BC. Register A indicates type of move desired. A zero value causes a direct move within the alternate bank; a

value less than zero causes a move from the Main RAM to the alternate, while a number greater than zero causes a move from the alternate bank to the Main System RAM.

19) F036 - Return Disk Map Table Address.
The address of the Logical to Physical Disk Mapping Table is returned in registers HL. If register H is non-zero on entry, the table address is stored in the integer variable pointed to by HL. This allows easy access by high level programming languages. The table consists of two sections. The first section contains 16 two byte entries, one for each logical CP/M drive. The first byte of each pair indicates which physical disk driver to activate for an I/O request, the second specifies which physical unit within that physical driver to access. These byte pairs may be carefully rearranged with other byte pairs in the table. They may even be removed or overwritten, but they must not be duplicated elsewhere in the table. The second part of the table holds the addresses of eight physical disk driver entry points. By convention, the driver number 0 always returns an error. It is used to force Select errors on unidentified logical drives. Driver number 1 controls WD-1797-02 floppy disk system, while driver number 2 manages the SA-1403 combination rigid/floppy disk system. Additional virtual disk drivers linked into this table, with appropriate values in the first section, may be accessed through the normal CP/M disk I/O facilities.

20) F039 - Return Address of Time of Day Locations.
This entry must be used to gain access to the timer variables maintained by the BOS. As above, if register H is non-zero on entry, it is used as an address of an integer variable in which to store the result. In any case, HL holds the timer address on exit. The returned address points to the following structure:

```
   Milsec:  ds  2   ;Location incremented by CTC1 interrupt
            ds  2   ;(unused)
   Ticker:  ds  2   ;Increments once per second
   Steprt:  ds  1   ;WD1797 step rate
   Motor:   ds  1   ;Disk Motor / Select timeout (1HZ)
HL- day:    ds  1   ;01-31
   month:   ds  1   ;01-12
   year:    ds  1   ;80-99
   hour:    ds  1   ;00-23
   minute:  ds  1   ;00-59
   second:  ds  1   ;00-59
   1 inbuf: ds  80  ;CRT gold mine
```

21) F03C - Return Configuration Status.
This function returns in register HL, or in the variable pointed to by HL, if H is non-zero, the current 820-II configuration. This function should be used to find out what kind of disk system is present, the current keyboard mask state, or other variable information concerning the 820-II. Only three bytes are currently defined, but more may be added in later releases. The status is as follows:

H = 00000000
L = kdf00000

k = Keyboard bit-8 mask
d = Rigid disk present
f = 5.25" floppies present

22) F03F - SIO-B Output Ready Status.
Returns FF if ready to transmit, 00 if not.

23) F042 - Get or Set Configurable Data.
This entry used by the CONFIGUR program, not by users.

24) F045 - Start Screen Print.
Initiates background screen print. Don't change the screen during printing, or results will be strange.

25) F048 - User Accessible 1 Second Interrupt.
This is actually another exit point. It is called by the real time clock interrupt service routine once each second. The user <u>must</u> follow the rules of interrupt service. Only registers HL and AF may be used, any others must be saved/restored on the 5 level stack provided. You must terminate with a RET instruction, or a jump to the address in this vector prior to patching your address in.

Display Control Codes

The video display may be controlled by various control codes and escape sequences to perform sophisticated screen manipulations. The Xerox 820-II responds to the following codes when CP/M is running and they are sent to the CRTOUT routines in the BOS.

Control Sequences

Code (hex)	Keyboard Sequence	Function
05	CTRL+E	Set cursor character as next character
06	CTRL+F	Restore previous attribute mode
07	CTRL+G	Bell
08	CTRL+H	Backspace or cursor left
09	CTRL+I	Horizontal tab
0A	CTRL+J	Line feed or cursor down
0B	CTRL+K	Cursor up
0C	CTRL+L	Cursor right
0D	CTRL+M	Carriage return
11	CTRL+Q	Clear to end of screen
18	CTRL+X	Clear to end of line
1A	CTRL+Z	Clear screen and home cursor
1B	CTRL+[	Escape
1E	HELP	Home Cursor
1F	CTRL+-	Display next charcter direct

Escape Sequences

Code (hex)	ESC followed by	Function
28	(	Disable attribute display
29	)	Enable attribute display
2A	*	Clear screen
30	0	Pass 7-bit keyboard data
31	1	Pass 8-bit keyboard data
34	4	Set blink attribute mode
35	5	Set graphics attribute mode
36	6	Set blink attribute mode
37	7	Set inverse video attribute mode
38	8	Set low intensity attribute mode
3D	=	XY cursor position leadin
45	E	Line insert
51	Q	Character insert
52	R	Line delete
57	W	Character delete

Display Code Description

The display control codes of the Xerox 820-II are completely compatible with the Xerox 820 with several advanced editing features added. The following summarizes the effect of each of the display codes:

Control Codes

"CTRL E" Set cursor character. After receiving this code the next character is interpreted as the code to be used as the cursor character. Only codes between 0 and 20 (hex) will be accepted. The normal cursor character is 2. The "space" (hex) character is a special case used to eliminate the display of a cursor. This is useful for displaying a screen without a large visible moving cursor for special effects.

"CTRL F" Restore previous attribute mode. Whenever the attribute mode is changed, the previous mode is remembered by the Xerox 820-II. In this way a program can set its own attributes mode for its unique display requirements, and then restore the original mode that was in effect before the program was run. Since the user may set his own default attribute mode with the CONFIGUR program in CP/M, it is desirable to restore this mode after it has been temporarily changed.

"CTRL G" Bell. This code will sound a short tone to alert the operator.

"CTRL H" Backspace or cursor left. Moves the cursor one column position to the left without altering the character under the cursor.

"CTRL I" Horizontal tab. Moves the cursor to the next tab stop. Tabs are preset for every eighth column.

"CTRL J" Line feed or cursor down.

"CTRL K" Cursor up. Moves the cursor up one row without effecting the current column position.

"CTRL L" Cursor right. Moves the cursor one column position to the right without altering the character under the cursor.

"CTRL M" Carriage return. Returns the cursor to the first column position of the current row.

"CTRL Q" Clear to end of screen. Clears all characters to spaces beginning with the current cursor position to the end of the screen. The position of the cursor remains unchanged. Characters before the cursor remain unchanged.

"CTRL X" Clear to end of line. Clears all characters from the current cursor position to the end of the current line to spaces. The cursor position is unchanged. Characters before the cursor are unchanged.

"CTRL Z" Clear screen and home cursor. Clears the entire screen of spaces and places the cursor in the home position (column 0, row 0).

"CTRL [" or "ESC" Escape. The first character of an escape sequence. These sequences are explained below.

"CTRL^" Home Cursor. Moves the cursor to the home position (column 0 row 0) without otherwise affecting the screen display. (HELP key produces 1E hex code)

"CTRL -" Display next character direct. After receiving this display code, the next character is displayed directly on the screen without interpreting it as a special display function code. This code is usually used to display control characters that are not normally displayed by the BOS.

Escape Sequences

"ESC (" Disable attributes display. Will cause all succeeding characters displayed on the screen to unconditionally have the upper bit reset, so that the selected attribute mode will not be displayed. Display will continue in this mode until changed by the "ESC)" sequence code.

"ESC)" Enable attribute display. Setting this mode will cause all following characters displayed on the screen to unconditionally have the upper bit set, thereby causing the selected attribute mode to be displayed. This mode will continue in effect until the "ESC (" mode disables it.

"ESC *" Clear screen. This function is the same as the CTRL + Z function, and clears the screen to spaces with the cursor at the home position.

"ESC 0" Sets BOS to pass only 7 bits of data from the keyboard. This is the default setting when power-on or reset is applied, and is compatible with the earlier Xerox 820. This mode of operation does not allow many of the unique codes generated by the keyboard to be used by applications software. A corollary effect is also automatically engaged in the 7-bit mode. Only 7 bits of data will be passed to the video display screen. ASCII characters with the upper bit set will normally cause one of the four attributes to be displayed (blink, lowlight, inverse video, or graphics characters). The "ESC 0" mode prevents this sometimes undesired feature.

"ESC 1" Sets BOS to pass the upper bit of data from the keyboard. Using the CTRL key in conjunction with certain keys will set the upper (eighth) bit of that key, which allows these codes to be processed as special function keys by applications programs. The following keys will produce unique codes with the upper bit set when they are typed in conjunction with the CTRL key.

"ESC 4" or "ESC 6" Set blinking attribute mode. This code will not actually begin displaying blinking characters on the screen. An "ESC)" sequence is used to enable the display of the attribute characters, or storing characters on the screen with the upper bit set, as described above. Thus, different attribute modes can be selected without affecting the screen display as long as there are NO characters on the screen with the upper bit set. If there ARE characters displayed on the screen with upper bit set, changing attribute modes will cause an IMMEDIATE change in the way the upper bit characters are displayed, depending on the attribute mode selected.

"ESC 5" Set graphic character attribute mode. This code will not actually begin displaying graphics characters on the screen. An "ESC)" sequence is used to enable the display of the attribute characters, or storing characters on the screen with the upper bit set, as described above. Thus, different attribute modes can be selected without affecting the screen display as long as there are NO characters on the screen with the upper bit set. If there ARE characters displayed on the screen with upper bit set, changing attribute modes will cause an IMMEDIATE change in the way the upper bit characters are displayed, depending on the attribute mode selected.

"ESC 7" Set inverse video attribute mode. This code will not actually begin displaying inverse video characters on the screen. An "ESC)" sequence is used to enable the display of the attribute characters, or storing characters on the screen with the upper bit set, as described above. Thus, different attribute modes can be selected without affecting the screen display as long as there are NO characters on the screen with the upper bit set. If there ARE characters displayed on the screen with upper bit set, changing attribute modes will cause an IMMEDIATE change in the way the upper bit characters are displayed, depending on the attribute mode selected.

"ESC 8" Set low intensity attribute mode. This code will not actually begin displaying characters on the screen in low intensity. An "ESC)" sequence is used to enable the display of the attribute characters, or storing characters on the screen with the upper bit set, as described above. Thus, different attribute modes can be selected without affecting the screen display as long as there are NO characters on the screen with the upper bit set. If there ARE characters displayed on the screen with the upper bit set, changing attribute modes will cause an IMMEDIATE change in the way these upper bit characters are displayed, depending on the attribute mode selected.

It should be noted that the low intensity mode is the DEFAULT attribute mode. This is because the D(isplay memory) command in BOS will store the ASCII character representation of the hex code so that more information about the dump is available. The low intensity mode is more suitable for this type of display. When CP/M is loaded, the CONFIGUR program allows the user to select his own default attribute mode.

"ESC =" Position the cursor to the location indicated by the following two row and column codes. The "home" position is designated as row 0, column 0. An offset of 20h must be added to the X and Y position codes. The positioning formula is:

ESC = (Y+20h) (X+20h)

where legal Y (row) values are between 0 and 79. If the column or row position codes exceed the normal 80 column or 24 row boundaries, the cursor will not be positioned to the illegal coordinate.

"ESC E" Line insert. Will move the entire line on which the cursor resides down one line, filling the cursor line with spaces, and causing the line on the bottom of the screen to disappear. (It is actually moved to the internal command line buffer for the monitor so that applications programs wishing to preserve the bottom line are able to do so.) The actual position of the cursor will not change.

"ESC Q" Character insert. Will insert a space at the current cursor position, causing the character under the cursor and all characters after the cursor to be shifted one position to the right. The last character on the line will disappear. The cursor position will remain unchanged and the character under the cursor will be the inserted space. No other lines will be affected. The character that was "lost" at the end of the line will actually be placed into the A register and the HL register will be pointing to the current cursor position upon return from the Fast CRT jump vector entry point (0F00Fh) so that applications programs can preserve this character.

"ESC R" Line delete. Similar to the line insert function except that the line on which the cursor resides will be deleted from the screen (and moved to the line buffer as described above), and all lines below it will be moved up one line. The position of the cursor will be unchanged.

"ESC W" Character delete. This function will delete the character under the cursor and cause all characters to the right of the cursor to move one position to the left. The last character position of the line will be replaced by a space. The cursor position will be unchanged and the character under the cursor will now be the character that was to the immediate right of the cursor before the character delete operation. The deleted character will be placed into the A register and the HL register will be pointing to the current cursor position upon return from the Fast CRT jump vector entry point (0F00Fh) so that applications programs can preserve this character.

"ESC T" Clear to End of Line. (Same as CTRL + X)

"ESC Y" Clear to End of Screen. (Same as CTRL + Q)

Numeric Pad

Key	ASCII Code (hexadecimal value)	
0	B0h	
1	B1h	
2	B2h	
3	B3h	
4	B4h	
5	B5h	
6	B6h	
7	B7h	
8	B8h	
9	B9h	
period	AEh	
plus sign	ABh	
minus sign	ADh	
up arrow	81h	
down arrow	82h	
right arrow	83h	
left arrow	84h	
line feed	8Ah	
Esc	9Bh	(Special key reserved for program abort)
Del	FFh	

Main Keyboard

Key	ASCII Code (hexadecimal value)	
Help	9Eh	(Special key reserved for Screen Print function)
1	91h	
2	92h	
3	93h	
4	94h	
5	95h	
6	96h	
7	97h	
8	98h	
9	99h	
0	90h	
=	9Ah	
backspace	88h	
tab	89h	
return	9Dh	

This is a total of 33 additional function keys that are available for applications programs when the ESC 1 mode is enabled. The natural result of this mode is that characters passed to the video display that have the upper bit set will be stored on the screen with the upper bit on, which causes the selected attribute mode to be enabled.

MONITOR RESIDENT I/O DRIVER FUNCTIONS

This section describes the facilities available in the Xerox 820-II monitor for controlling the input/output resources of the Xerox 820-II.

1) INTERRUPT PROCESSING

The Xerox 820-II monitor takes advantage of the powerful interrupt handling capabilities of the Z-80 microprocessor. Interrupts are utilized in the I/O drivers for the console keyboard input, the real-time clock and the floppy disk controller. All necessary initialization tasks and interrupt service routines for these devices are contained in the monitor.

For the most part, the operation of the interrupt mechanism should be transparent to most applications programs that will run under Xerox 820-II. A few precautions must be taken however, to ensure that user written software does not adversely effect the operation of the system. The following list describes the major hazards to the interrupt system:

- Interrupts should not be disabled permanently by user code, as this will lock-up the console input and real-time-clock routines.
- The Z-80 'I' register should never be altered. Doing so is GUARANTEED to crash the system.
- The CPU operates in Z-80 interrupt mode 2 and should not be switched to either of the other two interrupt modes.
- Adequate stack space must be reserved in user programs to allow at least one level of stack for interrupt return addresses. Use of the stack pointer for 'trick' programming purposes is highly discouraged for the same reason.

The monitor initializes the Z-80 'I' register to point to the system interrupt vector table at location FF00 to FF1F hex. This table contains pre-assigned vector locations for all the peripheral devices on Xerox 820-II, including those that are not used by any built-in functions in Xerox 820-II.

2) MEMORY MAPPED VIDEO DISPLAY

The Xerox 820-II single-board computer is equipped with a built-in 80 character by 24 line CRT display controller, for use with an external video monitor as the system console output device. The refresh memory for the CRT is bank switchable from the system's 64K byte memory space and includes a hardware address translation circuit for high speed scrolling.

The Xerox 820-II monitor contains an output driver routine for the CRT that emulates the characteristics of a typical stand-alone video terminal. All character codes between 00 and 7F hex are directly displayable on the screen. Characters are formed in a 5 x 7 dot matrix.

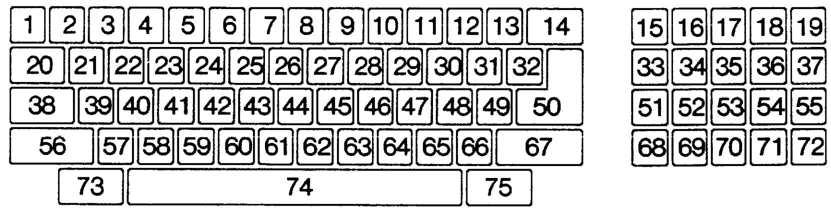
DISPLAY CHARACTER CODES

This table shows the code for each character to be displayed by the system. Each character is defined by a unique eight bit code which is represented by a hexadecimal code 'XY' where X represents the 4 most significant bits of the code and Y represents the 4 least significant bits of the code.

There are a total of 128 characters in the font set. Therefore, Y represents a hexadecimal number from 0 to F, and X represents a hexadecimal number from 0 to 7. Therefore, the complete font set is defined by codes from 00 to 7F. If the most significant bit of the code is set to '1', then the complete font set is duplicated with the blink attribute set. The blinking set of characters is then defined by codes from 80 to FF.

X \ Y	Ø	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
Ø	□	¢	█		§	½	¼	±	⇤	↑	↓	→	←	▒	®	↦
1	³	²	°	▬	⤓	¶	⤒	µ	✚	▌	↵	↵	©	⇥	⇤	
2		!	"	#	$	%	&	'	(	)	*	+	,	-	.	/
3	Ø	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
4	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
5	P	Q	R	S	T	U	V	W	X	Y	Z	[	\	]	^	_
6	`	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
7	p	q	r	s	t	u	v	w	x	y	z	{	|	}	~	Tm

KEY STATION NUMBERING AND KEY CODES



KEY NAME	KEY #	UNSHIFTED	SHIFTED	CONTROL
HELP	01	1E	1E	9E
1	02	31	21	91
2	03	32	40	92
3	04	33	23	93
4	05	34	24	94
5	06	35	25	95
6	07	36	5E	96
7	08	37	26	97
8	09	38	2A	98
9	10	39	28	99
0	11	30	29	90
MINUS	12	2D	5F	1F
EQUAL	13	3D	2B	9A
BACKSPACE	14	08	08	88
DELETE	15	7F	7F	FF
- (PAD)	16	2D	2D	AD
7 (PAD)	17	37	37	B7
8 (PAD)	18	38	38	B8
9 (PAD)	19	39	39	B9
TAB	20	09	09	89
Q	21	71	51	11
W	22	77	57	17
E	23	65	45	05
R	24	72	52	12
T	25	74	54	14
Y	26	79	59	19
U	27	75	55	15
I	28	69	49	09
O	29	6F	4F	0F
P	30	70	50	10
[	31	5B	7B	1B
]	32	5D	7D	1D
ESC	33	1B	1B	9B
+ (PAD)	34	2B	2B	AB
4 (PAD)	35	34	34	B4
5 (PAD)	36	35	35	B5
6 (PAD)	37	36	36	B6
LOCK	38	---	FUNCTION KEY	---

KEY NAME	KEY #	UNSHIFTED	SHIFTED	CONTROL
A	39	61	41	01
S	40	73	53	13
D	41	64	44	04
F	42	66	46	06
G	43	67	47	07
H	44	68	48	08
J	45	6A	4A	0A
K	46	6B	4B	0B
L	47	6C	4C	0C
SEMICOLON	48	3B	3A	7E
APOSTROPHE	49	27	22	60
RETURN	50	0D	0D	8D
LINEFEED	51	0A	0A	8A
UP ARROW	52	01	01	81
1 (PAD)	53	31	31	B1
2 (PAD)	54	32	32	B2
3 (PAD)	55	33	33	B3
L SHIFT	56	---	FUNCTION KEY	---
Z	57	7A	5A	1A
X	58	78	58	18
C	59	63	43	03
V	60	76	56	16
B	61	62	42	02
N	62	6E	4E	0E
M	63	6D	4D	0D
COMMA	64	2C	3C	1C
PERIOD	65	2E	3E	7C
SLASH	66	2F	3F	5C
R. SHIFT	67	---	FUNCTION KEY	---
L. ARROW	68	04	04	84
D. ARROW	69	02	02	82
R. ARROW	70	03	03	83
0 (PAD)	71	30	30	B0
. (PAD)	72	2E	2E	AE
L. CTRL	73	---	FUNCTION KEY	---
SPACE BAR	74	20	20	00
R. CTRL	75	---	FUNCTION KEY	---

NOTE: The codes listed above are the actual hex codes produced by the keyboard. The keyboard input routine in the monitor, sets bit 7 of all characters to ∅. When a CTRL + DEL is entered, the keyboard will output **FF** (hex) but the keyboard input routine converts this to **7F** (hex).

3) **PARALLEL KEYBOARD INPUT**

A parallel keyboard interface is provided on Xerox 820-II for systems that will use the built-in keyboard and CRT display as the console I/O device. This interface is designed to connect to an ASCII encoded keyboard with 8 bits of parallel data and a key-pressed strobe.

The monitor contains an interrupt driven input handler for the keyboard that maintains a 16 character deep FIFO buffer for input data. This makes it possible to do a considerable amount of typing ahead without any input characters being lost. If characters are typed while disk access is going on, they may be lost because the disk routines lock out all lower priority interrupts. Any characters received when the FIFO is full will also be lost.

4) **DISK INTERFACE**

The 820-II (with floppy disk drives) has an edge-card connector for interfacing to a "baby board" for disk interface. The 820-II will be equipped with a "baby board" for controlling up to two Shugart compatible 8" drives, or two 5¼" drives. The interface hardware on the 820-II "baby board" (floppy disk drives) is based on a western Digital 1797 Floppy Disk Controller chip along with extra TTL Support circuitry to provide buffering, drive select, and data separator functions.

The 820-II (with a rigid disk drive) will be equipped with a "baby board" for interfacing to a Shugart 1403D Controller which in turn will control up to four drives of various mixes of Shugart compatible 8" floppy and 8" rigid disk drives. The interface hardware on the 820-II "baby board" is based on a Z80A PIO chip.

The monitor contains a complete I/O driver package for the disk controller. Linkage to the disk I/O routines in the monitor is provided by a set of sub-routine entry points described later in this manual. The basic functions available are: drive select, restore, seek track, read sector, and write sector. The user can also specify the track-to-track seek stepping rate, and the sector record length.

All disk functions are verified upon completion, with the final status being returned in the A register. If the command was executed successfully, then A will contain all zeros on return, otherwise it will contain an error status byte as described above under the console monitor 'R' command. The disk drive routines will attempt to recover from any disk I/O errors that occur, so it is generally not necessary for user written programs to try to re-execute commands that fail the first time.

5) **SERIAL INPUT/OUTPUT**

The computer has provisions for two completely independent RS-232 serial ports that can be used to interface to printers, CRT terminals and data communications equipment.

6) **REAL TIME CLOCK**

The computer has a Z-80 CTC device that can be used to generate the timebase for interrupt driven timers, real-time clocks, and other time keeping functions. The monitor will initialize CTC channels 2 and 3 to interrupt the processor once a second. Channels 0 and 1 of the CTC are not initialized and can be used for other purposes.

The one second interrupt from the CTC is utilized by the monitor's disk I/O routines to implement the disk motor turn-off function.

7) **PARALLEL I/O OPTION**

A Z-80 PIO chip has been included on the Xerox 820-II IP for general purpose I/O interfacings. This device is completely unused by any built-in functions. The PIO contains two independent 8-bit parallel I/O ports that can be used to interface to printers, ROM programmers, analog converters, other computers, or just about anything else imaginable. Those interested in using the PIO should consult the schematic drawings for any needed hardware interfacing details. Data about programming the PIO can be found in most Z-80 applications manuals.

DISK FORMAT

DISK PARAMETERS

	5¼"				8"			
	SSSD	DSSD	SSDD	DSDD	SSSD	DSSD	SSDD	DSSD
Tracks	1-39	1-79	1-39	1-79	1-76	1-153	1-76	1-153
Sectors/ Track	18	18	17	17	26	26	26	26
Bytes/ Sector	128	128	256	256	128	128	256	256
Reserved Tracks	3	3	3	3	2	2	2	2
* Disk Capacity	82K	172K	155K	322K	241K	490K	482K	980K

DISK FORMAT (floppy disks)

The system is equipped with two (2) compatible Shugart 5¼" drives, or two 8" Shugart drives. The disks for the 5¼" drives are initialized in a CROMEMCO format, and the disks for the 8" drives are initialized in an IBM 3740 format.

A format is divided into three (3) parts; field A, field B, and field C. Field A is written at the start of each track known as the preamble. Field B is written once for each sector which consists of a gap between sectors, ID fields, and a data field. Field C is written at the end of each track and is known as a postamble.

TRACK 0 FORMAT

Track 0 is written in single density and in the following format:

PARAMETER	8"	5¼"
Tracks	0	0
Sectors	26	18
Bytes/Sector	128	128

5¼" Format - Track 0

Number of Bytes	Hex Value of Bytes	Comment
16	FF	Preamble on Gap 4A
4	00	Gap 3
1	FE	ID Address Mark
1	XX	Track #
1	00	
1	XX	Sector #
1	00	
* 1	F7	Generate CRC (2)
11	FF	Gap 2
6	00	
1	FB	Data Address Mark
128	E5	Data Field
1	F7	Generate CRC (2)
8	FF	Gap 3
101	FF	Postamble Gap 4B

* Repeated for # sectors per track.

8" Format - Track 0

8" data architecture is similar to IBM 3740 format.

Number of Bytes	Hex Value of Bytes	Comment
28	FF	Preamble - Write at the
6	00	start of each track
1	FC	
26	FF	
6	00	Gap 3
1	FE	ID Address Mark
1	XX	Track #
1	00	
1	XX	Sector #
* 1	00	
1	F7	Generate CRC
11	FF	Gap 2
6	00	
1	FB	Data Address Mark
128	E5	Data Field
1	F7	Generate CRC
27	FF	Gap 3
247	FF	Postamble Gap 4B

* Repeated for # of sectors per track.

OTHER TRACKS

The disks are initialized in the following format:

PARAMETER	8"	5¼"
Tracks	1-76	1-39
Sectors	26	17
Bytes/Sector	256	256
Tracks reserved for OS*	2	3

Reserved tracks for 8" - Track 0, 1

Reserved tracks for 5¼" - Track 0, 1, and 2

5¼" Format - Tracks 1 to 39

Data architecture is IBM a modified System 34 format.

	Number of Bytes	Hex Value of Bytes	Comment
	50	4E	Gap 1
	12	OO	Sync Field
	3	F5	ID Address Mark
	1	FE	ID Mark
	1	XX	Track
	1	XX	Head#
	1	XX	Sector
*	1	01	Sector Length ID
	1	F7	CRC Generates 2 Bytes
	22	4E	Gap 2
	12	OO	Sync Field
	3	F5	Data Address Mark
	1	FB	Data Mark
	256	E5	Data
	1	F7	CRC Generates 2 Bytes
	32	4E	Gap 3 Between Sectors
	284	4E	Postamble

8" Format - Tracks 1 to 76

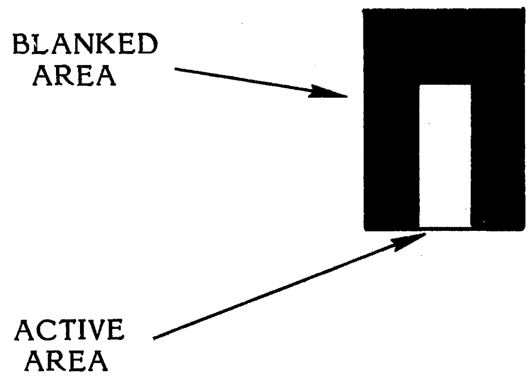
Data architecture is IBM 34 format.

Number of Bytes	Hex Value of Bytes	Comment
80	4E	Gap 4A
12	OO	Sync Field
3	F6	Index Address Mark
1	FC	Index Mark
50	4E	Gap 1
* 12	OO	Sync Field
* 3	F5	ID Address Mark
* 1	FE	ID Mark
* 1	XX	Track
* 1	XX	Head#
* 1	XX	Sector
* 1	01	Sector Length ID
* 1	F7	CRC Generates 2 Bytes
* 22	4E	Gap 2
* 12	OO	Sync Field
* 3	F5	Data Address Mark
* 1	FB	Data Mark
* 256	E5	Data
* 1	F7	CRC Generates 2 Bytes
* 54	4E	Gap 3 Between Sectors
600	4E	Postamble

* Repeated for # of sectors per track.

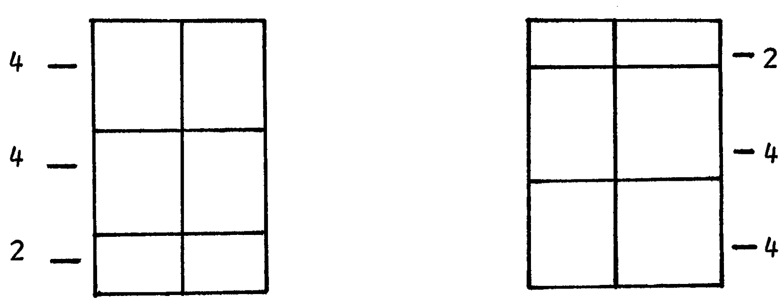
GRAPHICS

The 820-II Display Controller is based on displaying characters within a 7 x 10 character cell (7 dots horizontal by 10 scan lines vertical). To guarantee spaces between characters, one dot on each side of the character cell is blanked by hardware. Also, to guarantee spaces between character lines, the top two scan lines are blanked by hardware. This gives an actual active character size of 5 dots horizontal by 8 scan lines vertical as shown below:

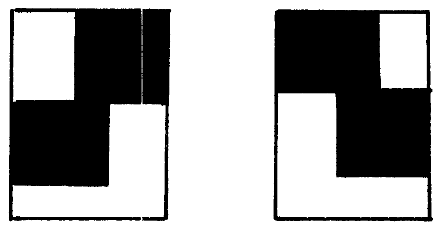


For Business Graphics, the hardware is configured to eliminate the automatic blanking and allow continuous lines both horizontal and vertical. However, the Display Controller is still based on displaying a character within a 7 x 10 character cell. Since the 820-II is an 8 bit system and the controller design and available refresh memory allows only one byte per character, the maximum number of unique characters that can be defined by any 8 bits is 256. Since the standard 820-II text font set contains 128 characters, the limit on unique characters for graphics that can be displayed together with text is 128.

The character set for Business Graphics divides the character cell into blocks of 4 dots horizontal by 4 scan lines vertical. Since the total number of scan lines per character is 10, the character set actually consists of two subsets of 4-4-2 and 2-4-4 as shown below:



Each subset divides the character cell into 6 parts requiring 64 possible combinations or unique characters. Therefore, the total number of unique characters for the complete graphics set is 128. With this character set, any combination of adjacent 4 x 4 blocks can be chosen. Also, at the character cell boundary the 4 x 4 blocks can be set vertical by 2 scan lines. Since the total number of horizontal dots per character cell is 7, there will be an overlap of one horizontal dot in the center of the character cell for diagonal blocks within the cell as shown below:



It should be noted that the above examples are shown inverted from actual display of 820-II. That is, the 820-II will display 4 x 4 white on black vs. the above examples which are shown 4 x 4 black on white.

It should be also noted that for the standard text font containing 128 unique characters defined by 7 bits, the eighth bit is used for the blink attribute. For Business Graphics, since both text characters and graphic characters can be displayed simultaneously, this requires all 8 bits to define the character, consequently the blink attribute is not available in graphics mode.

THEORY OF OPERATION — CENTRAL PROCESSOR

CLOCK GENERATOR

All the system clocks with the exception of the baud clock and the video dot clock are generated from a master oscillator operating at 16 Mhz. The 2 Mhz (1 Mhz for 5¼") clock for the disk controller is generated from the 16 Mhz clock by a divide by 8 counter.

The 4 Mhz processor clock is generated by dividing the master 16 Mhz clock by 4 with binary counter. The column address strobe "CAS", and the address multiplexer control "MUX", are derived from the 16 Mhz clock. When memory request "MREQ" is low and refresh "RFSH" is high generation of "CAS" and "MUX" is enabled.

RESET CONTROLLER

Two types of reset take place on the board. Power on reset is detected and conditioned by part of hex schmitt inverter. The pushbutton reset is also conditioned by a part of hex schmitt inverter.

PORT ADDRESS DECODING

Octal decoder is used to select the appropriate I/O device based on the binary value of the address bits A2, A3, & A4. When A7 is low and "M1R" is high, a low on "IORQ" will cause the appropriate output of the decoder to go low, selecting the I/O device for a read or write operation.

I/O Port Assignments For Floppy Disk Option

PORT 0-3 = CHANNEL A BAUD RATE (WRITE ONLY)
PORT 4 = SIO CHANNEL A DATA
PORT 5 = SIO CHANNEL B DATA
PORT 6 = SIO CHANNEL A CONTROL
PORT 7 = SIO CHANNEL B CONTROL
PORT 8 = GP PIO CHANNEL A DATA
PORT 9 = GP PIO CHANNEL A CONTROL
PORT A = GP PIO CHANNEL B DATA
PORT B = GP PIO CHANNEL B CONTROL
PORT C-F = CHANNEL B BAUD RATE (WRITE ONLY)
PORT 10 = FLOPPY DISK CONTROLLER STATUS/COMMAND REGISTER
PORT 11 = FLOPPY DISK CONTROLLER TRACK REGISTER
PORT 12 = FLOPPY DISK CONTROLLER SECTOR REGISTER
PORT 13 = FLOPPY DISK CONTROLLER DATA REGISTER
PORT 14-17 = CRT SCROLL REGISTER (WRITE ONLY)
PORT 18 = CTC CHANNEL 0
PORT 19 - CTC CHANNEL 1
PORT 1A = CTC CHANNEL 2
PORT 1B = CTC CHANNEL 3
PORT 1C = SYSTEM PIO CHANNEL A DATA
PORT 1D = SYSTEM PIO CHANNEL A CONTROL
PORT 1E = SYSTEM PIO CHANNEL B KEYBOARD DATA

PORT 1F = SYSTEM PIO CHANNEL B KEYBOARD CONTROL
PORT 20-27 = NOT USED/NOT AVAILABLE
PORT 28 = RESET AUDIBLE ALARM (WRITE ONLY)
PORT 29 = ACTIVATE AUDIBLE ALARM (WRITE ONLY)
PORT 2A-2F = NOT USED/NOT AVAILABLE
PORT 30 = SINGLE DENSITY (WRITE ONLY)
PORT 31 = DOUBLE DENSITY (WRITE ONLY)
PORT 32 = NOT USED/NOT AVAILABLE
PORT 33 = NOT USED/NOT AVAILABLE
PORT 34 = RESET CRT FONT GENERATOR TO ROM #1 (WRITE ONLY)
PORT 35 = SELECT CRT FONT GENERATOR TO ROM #2 (WRITE ONLY)
PORT 36 = SET LOLIGHT VIDEO MODE (WRITE ONLY)
PORT 37-67 = NOT USED/NOT AVAILABLE
PORT 68 = ASYNCHRONOUS COMMUNICATIONS (WRITE ONLY)
PORT 69 = SYNCHRONOUS COMMUNICATIONS (WRITE ONLY)

I/O Port Assignments For Fixed Drive Options

PORT 0-F = SAME AS FLOPPY DISK OPTION
PORT 10 = FIXED DISK PIO CHANNEL A DATA
PORT 11 = FIXED DISK CHANNEL A CONTROL
PORT 12 = FIXED DISK CHANNEL B DATA
PORT 13 = FIXED DISK CHANNEL B CONTROL
PORT 14-2F = SAME AS FLOPPY DISK OPTION
PORT 30-33 = NOT USED/NOT AVAILABLE
PORT 34-69 = SAME AS FLOPPY DISK OPTION

FLOPPY DISK TRANSFER SYNCHRONIZATION

In order to successfully execute the high speed data transfers between the processor and the disk controller, the fast Z-80 non-maskable interrupt "NMI" response was employed. During reads and writes to and from the disk controller, the data at memory location 66 hex is retrieved and stored. This location is overwritten with a RETURN instruction. After this setup is accomplished the processor executes a HALT instruction. When the processor is in a HALT condition, a DATA REQUEST (DRQ) or an INTERRUPT REQUEST (IRQ) from the disk controller will cause a non-maskable interrupt to be generated. The processor then executes the RETURN instruction at 66 hex and returns to transfer the data to or from the disk controller. When the 128 byte (or 256 bytes in double density) transfer is complete the old data is restored and the processor resumes normal operation. This hardware assistance obviated the necessity for a DMA device by eliminating the disk controller "DRQ" status test.

CRT DISPLAY CONTROLLER

VIDEO SCROLLING

In order to eliminate the delay associated with software scrolling, hardware assistance was employed. For ease of understanding, the CRT RAM resides from 3000 hex to 3FFF hex. Writing into the scroll register adds an offset to the line address developed by the line counter. The net effect is similar to the rotation of a cylinder whose axis is horizontal and perpendicular to the line of shift. The amount of rotation is determined by the magnitude of the number contained in the scroll register. For instance, an offset of zero puts the data at location 3000 hex (of the CRT memory) at the bottom of the screen. If the offset was one, the data at 3000 hex would be displayed on the line next to the bottom. An offset of seventeen hex (23 decimal) puts the data at location 3000 hex at the top of the screen.

VIDEO RAM ADDRESSING

If the processor is doing a read or write to video RAM "CRTCE" (CRT memory access enable) will go low. When "CRTCE" goes low, the address from the processor is selected instead of the address generated by the counter chain. This gives the processor access to the video RAM for read or write operations.

CPU ACCESS OF VIDEO RAM

During read or write operations involving the video RAM and the CPU, "CRTCE" will go low. When "CRTCE" goes low the processor address bus is selected as the address source for the video RAM. A low on "CRTCE" is also used as a term in the direction control logic for data bus access. During a processor read operation, data from the video RAM at the specified address is allowed onto the processor data bus. During a processor write operation, data from the processor is written to the video RAM at the specified address.

VIDEO GENERATION

While in the display mode, ASCII data from the video RAM and scan address data are used to select the proper dot patterns from the character generator. The character generator contains 1 font pattern of 128 characters.

DISPLAY BLANKING

The display is blanked during horizontal retrace, vertical retrace, CPU access, and decode of scan counts 8 and 9. Blanking is accomplished by disabling the character generator.

64 K RAM AND BANK SWITCHING

REFRESH

During the refresh cycle, the Z-80 places the refresh address on the lower 7 bits of the address bus. When this address is stable in the RAM array, the "RFSH" pin on the Z-80 goes low. The active low "RFSH" generates an "RAS" on all RAMs. An active "RFSH" disables the generation of both "CAS" and "MUX" .

BANK SWITCHING

Bit 7 of port 1C hex is the bank switch control. When the output is high, the ROMs and the CRT display appear in the lower 16K block, and the second 16K block is reserved for memory expansion via the bus access slot. When bit 7 of port 1C hex is low, all the 64K RAM is available to the processor.

CTC

The CTC resides at ports 18 hex through 1B hex. All the inputs and outputs associated with the CTC are available to the user. Refer to the strapping option section for pin assignments.

SYSTEM PIO

The system PIO resides at ports 1C hex through 1F hex. The "A" side of the system PIO controls the floppy disk drive select, bank switching, sensing keyboard data available (for polled keyboard applications), and an uncommitted user definable I/O bit. The bit allocations are as follows:

BIT 0 = DVSEL 1 (820-II), LOW (820-II Rigid Disk)
BIT 1 = DVSEL 2 (820-II), UNUSED (820-II Rigid Disk)
BIT 2 = SIDE (820-II), UNUSED (820-II Rigid Disk)
BIT 3 IS USED FOR KEYBOARD DATA AVAILABLE
BIT 4 IS 8"/5¼" DISK SELECT ((820-II), UNUSED (820-II Rigid Disk)
BIT 5 DOUBLE SIDED MEDIA (820-II), UNUSED (820-II)
BIT 6 CONTROLS DISPLAY CHARACTER SET
BIT 7 CONTROLS THE BANK SWITCHING (0=RAM)

GENERAL PURPOSE PIO AND SIO

The G.P. PIO provides the user with 16 bits of user definable input or output or a mix of input and output on nibble boundaries. The G.P. PIO resides at ports 08 hex -0B hex. The PIO will support all modes of interrupt supported by the Z-80. For detailed programming information refer to the Z-80 PIO data sheet. For applications information, refer to the strapping option section.

SIO

The Z-80 SIO supports two full channels of serial I/O with the capability of supporting full RS-232 protocol on both channels. In addition, the A side of the SIO can provide clocks to synchronous modems or receive clocks from the modem.

Channel A of the SIO can be configured to interface to a modem or a terminal. Refer to the strapping option sheets for detailed instructions. Refer to the SIO data sheet for programming information.

Channel B of the SIO is dedicated for printer operation and has no strapping options.

BAUD RATE GENERATOR

The COM 8116 provides the user with two programmable baud rate generators. Channel A baud rate resides at port 00 hex and is write only. Channel B baud rate resides at port 0C hex and is also write only. The programming procedure is as follows:

00 hex	=	50	Baud
01 hex	=	75	Baud
02 hex	=	110	Baud
03 hex	=	134.5	Baud
04 hex	=	150	Baud
05 hex	=	300	Baud
06 hex	=	600	Baud
07 hex	=	1200	Baud
08 hex	=	1800	Baud
09 hex	=	2000	Baud
0A hex	=	2400	Baud
0B hex	=	3600	Baud
0C hex	=	4800	Baud
0D hex	=	7200	Baud
0E hex	=	9600	Baud
0F hex	=	19.2	Kbaud

INTERRUPT STRUCTURES

All the Z-80 family devices on this board are capable of supporting mode 0, 1, and 2 interrupts. Mode 2 interrupts are used in the Monitor delivered with the system. The I register in an unmodified system is loaded with 0FF hex. The priority chain is organized high to low as follows:

SIO CHANNEL A
SIO CHANNEL B
SYSTEM PIO PORT A
SYSTEM PIO PORT B
GP PIO PORT A
GP PIO PORT B
CTC CHANNEL 0
CTC CHANNEL 1
CTC CHANNEL 2
CTC CHANNEL 3

GENERAL PURPOSE PIO STRAPPINGS (J11) AND PIN ASSIGNMENTS (J8)

J8

	1	2	
all odd	o	o	port A STROBE
numbered	o	o	port A READY
pins are	o	o	port A bit 0
grounded	o	o	port A bit 1
	o	o	port A bit 2
	o	o	port A bit 3
	o	o	port A bit 4
	o	o	port A bit 5
	o	o	port A bit 6
	o	o	port A bit 7
	o	o	port B READY
	o	o	port B STROBE
	o	o	port B bit 0
	o	o	port B bit 1
	o	o	port B bit 2
	o	o	port B bit 3
	o	o	port B bit 4
	o	o	port B bit 5
	o	o	port B bit 6
	o	o	port B bit 7
	39	40	

J11

	1	2	
PB6	o	o	PB4
	o	o	port B READY polarity
	o	o	port B lower direction
odd pins 3-17	o	o	port A READY polarity
are grounded	o	o	port A upper direction
	o	o	port B upper direction
	o	o	port A STROBE polarity
	o	o	port B STROBE polarity
	o	o	port A lower direction
+5V	o	o	+5V
	19	20	

CTC STRAPPING AND I/O ASSIGNMENTS (J10)

J10

	2	1	
SYSTEM CLOCK	o	o	CLOCK/TRIGGER 0
ZC-TO0	o	o	CLOCK/TRIGGER 1
ZC-TO1	o	o	CLOCK/TRIGGER 2
ZC-TO2	o	o	CLOCK/TRIGGER 3
	8	7	

VIDEO OUTPUT CONNECTOR PIN ASSIGNMENTS (J7)

J7

	6	1	
	o	o	
	o	o	
6-10 grounded	o	o	Vertical Sync
	o	o	Horizontal Sync
	o	o	Video
	10	5	

SERIAL I/O CONNECTOR PIN ASSIGNMENTS CHANNEL A (J4)

	J4		
	1	14	
PROTECTIVE GROUND	o	o	
TRANSMIT DATA	o	o	TRANSMIT CLOCK
RECEIVE DATA	o	o	
REQUEST TO SEND	o	o	RECEIVE CLOCK
CLEAR TO SEND	o	o	
DATA SET READY	o	o	
PROTECTIVE GROUND	o	o	DATA TERMINAL READY
CARRIER DETECT	o	o	
	o	o	
	o	o	
	o	o	
	o	o	
	o	25	
	13		

SERIAL I/O STRAPPING OPTIONS FOR CHANNEL A (J9)

Only channel A is capable of utilizing baud clocks from an external device or of providing baud clocks to an external device. When providing the baud clock to the external device the SIO must use the same clock source.

	J9		
	1	2	
	o	o	
	o	o	
	o	o	(M) RXD to Pin 3
	o	o	(T) TXD to Pin 2
	o	o	(M) TXD from Pin 2
	o	o	(T) RXD from Pin 3
	o	o	(M) CTS to Pin 5
	o	o	(T) RTS to Pin 4
	o	o	(M) RTS from Pin 4
	o	o	(T) CTS from Pin 5
	o	o	(M) DCD to Pin 8
	o	o	(T) DTR to Pin 20
	o	o	(M) DTR from Pin 20
	o	o	(T) DCD from Pin 8
Clock supplied to Modem as RX Clock	o	o	
Clock supplied to SIO with RX Clock	o	o	
Modem supplies SIO with RX Clock	o	o	
Clock supplied to SIO with TX Clock	o	o	
Modem supplies SIO with TX Clock	o	o	
Clock supplied to Modem with TX Clock	o	o	
	39	40	

DATA SET READY IS ACTIVE ON BOTH CHANNELS

Legend

(M) Indicates modem (Data Communications Equipment) function
(T) Indicates terminal (Data Terminal Equipment) function

For instance, exercising the (T) strap options will allow communication with a modem. Exercising the (M) strap options would allow communication with a terminal.

TXD = Transmitted Data
RXD = Received Data
RTS = Request to Send
CTS = Clear to Send
DTR = Data Terminal Ready
DCD = Data Carrier Detect

SERIAL I/O CONNECTOR PIN ASSIGNMENTS CHANNEL B (J3)

	J3		
	1	14	
Ground	o	o	
Receive Data	o	o	
Transmit Data	o	o	
Clear to Send	o	o	
Request to Send	o	o	
Data Set Ready	o	o	
Ground	o	o	Data Carrier Detect
Terminal Ready	o	o	
	o	o	
	o	o	
	o	o	
	o	o	
	o	25	
	13		

KEYBOARD CONNECTOR PIN ASSIGNMENTS (J2)

	J2	
	1	14
BIT 0	o	o
BIT 1	o	o
BIT 2	o	o
BIT 3	o	o
BIT 4	o	o
BIT 5	o	o
BIT 6	o	o
BIT 7	o	o
STROBE	o	o
	o	o
	o	o
	o	o
+5 volts	o	25
	13	

Pins 14-25 are all grounded

DISK DRIVE CONNECTOR PIN ASSIGNMENTS (J1)

RIGID	FLOPPY	J1	
		1	20
		o	o
NDI	8/5¼ Select	o	o
NMSG	Media	o	o
ND5	Index	o	o
ND6	Select 1	o	o
ND4	Select 2	o	o
ND7	Side	o	o
ND3	HDLD	o	o
NRST	Step In	o	o
NPB	Step	o	o
NC/D	Write Data	o	o
NREQ	Write	o	o
NI/O	TRK 00	o	o
NOT USED	Write Protect	o	o
NBSY	Read Data	o	o
NDZ	Low Current	o	o
NDØ	Ready	o	o
NSEL	+12VDC	o	o
NACK	+5VDC	o	o
		19	37

Pins 20-37 are all grounded

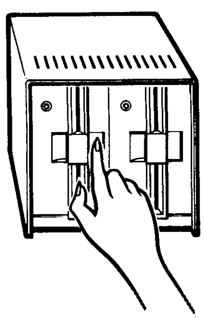
PREPARING TO RUN DIAGNOSTICS ON A NEW 820-II

Your Xerox 820-II Personal Computer comes with an 820-II Diagnostic Exerciser disk in the disk drive box. You can use this disk to check that your system is in proper working order. The disk will check the different components of the system and display a message if it finds a malfunction.

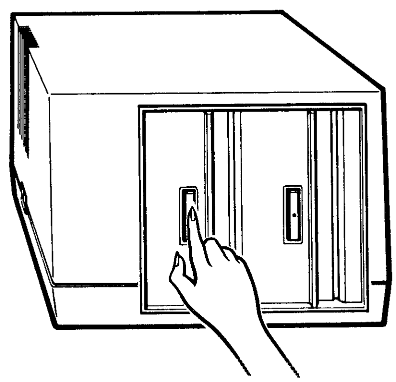
It's a good idea to check your system when you first install it. If you experience any problems while using the system, you can use the 820-II Diagnostic Exerciser disk to check the system and find out if it needs to be serviced (repaired).

If you have just installed a new Xerox 820-II, proceed with the steps below. If you've already used the Xerox 820-II, turn to page 8.

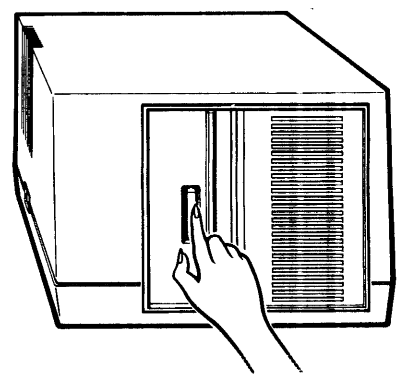
COMPARE your disk drive unit to the illustration below to determine what type of disk drive you have.



5¼" DUAL FLOPPY DISK DRIVE



8" DUAL FLOPPY DISK DRIVE

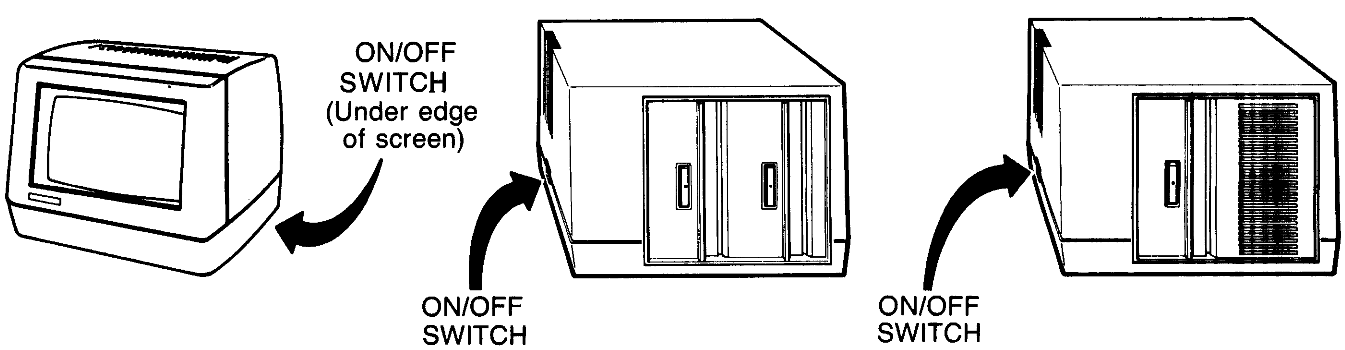


8" RIGID DISK DRIVE

OPEN the disk drives by pushing on the latch or button as shown above.

LOCATE the ON/OFF switch on the left side of the 8" disk drives (the 5¼" disk drive does not have an ON/OFF switch).

LOCATE the ON/OFF switch under the right side of the screen (toward the back).



TURN the 8" disk drives on -- if your system has 8" drives, they must be turned on for your system to operate.

TURN the screen on -- if it is already turned on, press the RESET button at the back of the screen or CTRL+ESC on the keyboard.

WAIT for your screen to display information in the left top corner of your screen. If it does not appear after a few seconds, try adjusting the brightness control (under the left edge of the screen) toward the front.

If nothing appears on the screen, your system needs servicing.

ADJUST the brightness control (under the left edge of the screen) for the best viewing.

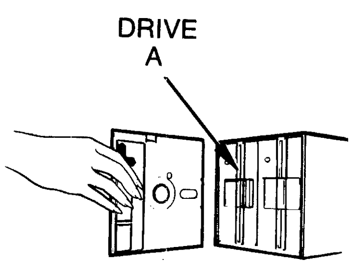
If you have dual floppy disk drives, continue with the steps below. If you have an 8" rigid disk, **turn to page 5.**

TO INITIALIZE A FLOPPY DISK

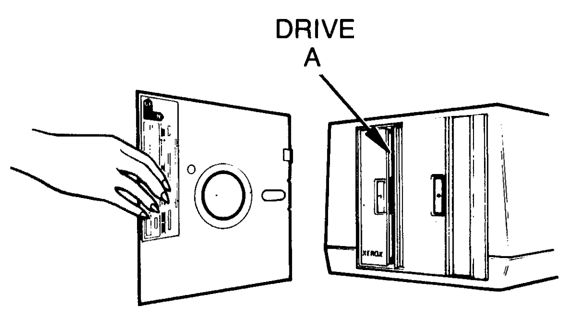
When you use the Xerox 820-II Diagnostic disk to check out a system that has dual floppy disk drives, you'll need to initialize a floppy disk to use for the test. An initialized disk is a disk that has been prepared for use.

FIND the CP/M disk (The CP/M disk comes packaged with the CP/M manuals).

INSERT the **CP/M** disk in the left drive (drive A) as shown below. Use the up and in arrows on the disks as guides.

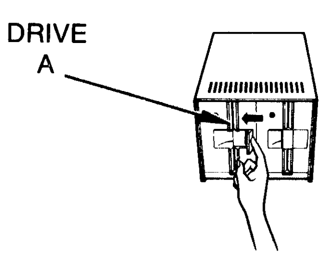


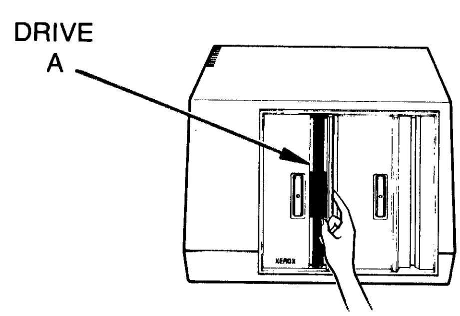
5¼" DISK DRIVES



8" DISK DRIVES

CLOSE the drive as shown below.



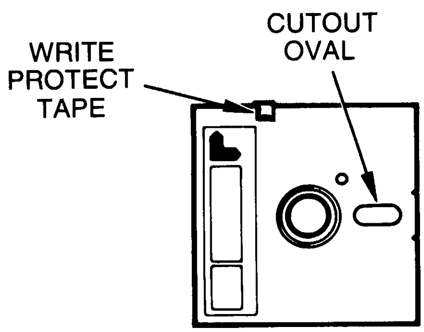


TYPE the letters **LA** and press the RETURN key on the keyboard. (If the screen displays the message "load error" it means the disk is in upside down. Remove the disk and resert it correctly. Press the RESET button on the back of the screen and repeat the above step.)

TYPE the word **INIT** and press the RETURN key.

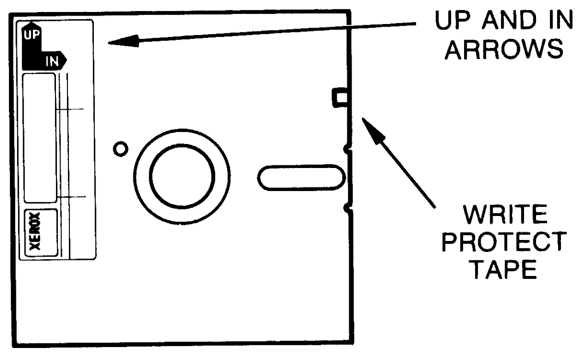
WAIT for the message "Enter physical disk drive to initialize (A or B)"

OBTAIN a new disk. Be sure the write protect tape is off the notch on the 5¼" disk, or the write protect tape is on the notch on the 8" disk. (Write protect tapes come packaged in the box of disks when you purchase them. If you don't have a new disk, a previously used one will do).



5¼" DISK

lus of di



8" DISK

HOLD the disk with the oval cutout toward the drive (as shown in the picture on the opposite page).

INSERT the new disk in the right drive (drive B) and close the door.

TYPE the letter **B**

The following message to be displayed on the screen:

```
Available disk formats are:

1)     Single Density, Single Side
2)     Single Density, Double Side
3)     Double Density, Single Side
4)     Double Density, Double Side

   Enter desired format, or  ESC  to re-enter selections
```

TYPE the number **3**

The following message is displayed on the screen:

```
Are you ready to ERASE (initialize) the disk in physical drive B (Y/N)?
```

TYPE the letter **Y**

WAIT several minutes for the following message to display on the screen:

```
        "0 Defective Sectors"

Touch any key to continue or  CTRL+C  to Exit
```

IF the disk has defective sectors indicated by a number other than 0 in front of the Defective Sectors message, do not use it - initialize another disk using the steps below:

- Replace the disk in drive B with another disk.
- Touch any key on the keyboard and follow the directions on the screen.
- Wait for the "0 Defective Sectors - Touch any key to continue or CTRL+C to Exit" message to display on the screen.

REMOVE both of the disks from the disk drives.

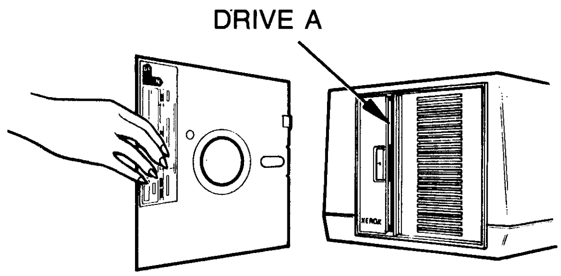
You're now ready to check out the system with the Diagnostic Exerciser disk. The instructions for running diagnostics start on page 8.

TO FORMAT (INITIALIZE) A RIGID DISK

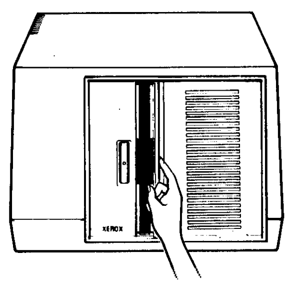
DO THIS PROCEDURE ONLY IF THE SYSTEM IS NEW. If the rigid drive has been used, this procedure will erase all stored data.

Before you use the diagnostic exerciser disk to check a system with a rigid disk drive, you'll need to format (initialize) the rigid disk before running the test.

INSERT the CP/M disk in the left disk drive (drive A).



CLOSE the drive as shown below.



TYPE the letters **LA** and press the RETURN key on the keyboard.

TYPE the letters **FMT** and press the RETURN key.

Result The following message will be displayed on the screen:

```
            Rigid Disk Initialization Utility VER. 0.00
               Copyright (C) 1982, XEROX Corporation

INITIALIZING Will ERASE ALL The DATA On The RIGID DISK
Touch  ANY KEY  To EXIT Or  RETURN  To CONTINUE
```

PRESS the **RETURN** key to continue.

Result The following message is displayed on the screen:

```
Are You SURE You Want To CONTINUE ? (Y/N)
```

TYPE the letter **Y** and press the RETURN key.

Result The following message is displayed on the screen:

```
Initializing Rigid Disk Track    0000
```

WAIT several minutes for the following message to be displayed on the screen:

```
Please run the Verify Disk Integrity section of the
Xerox Disk Backup and Maintenance System Utility to
identify any flawed sectors.

It is necessary for you to reload the system
Touch  ANY KEY  To RESET the machine
```

TOUCH any key on the keyboard.

TYPE the letters **LA** and press the RETURN key.

TYPE the word **BACKUP** and press the RETURN key.

Result The following menu is displayed on the screen:

```
                The Xerox Disk Backup and Maintenance System
                (C) 1982 Balcones Computer Corporation (P)
                All Rights Reserved Version 0.00  (Month)  1982

                    DISK BACKUP & MAINTENANCE MENU

          You have the following options available:

   (1)    List Directory
   (2)    Backup Files
   (3)    Replace Files
   (4)    Verify Disk Integrity
   (5)    Delete Files
   (6)    Exit to CP/M

          Please enter your choice: (_)
```

Note: When using the BACKUP Utility, you may want to increase the brightness of your screen using the brightness control located under the left edge of the screen.

TYPE the number **4** to select the Verify Disk Integrity option.

Result The following message is displayed on the screen:

```
                    VERIFY DISK INTEGRITY

   Verify which Disk: (_)
```

TYPE the letter **E** to verify disk (partition) E which is used by the diagnostic exerciser program to check the operation of the disk drives.

Result The following message is displayed on the screen:

```
   0000  Reading 0 to 1007 Blocks.  (Touch  ESC  at any time to exit.)
```

WAIT several minutes for the following message to be displayed on the screen:

```
   No errors detected.  Touch any key to exit.
```

IF an error is detected, TYPE the letter **Y** to automatically store the error (flawed area) in a special file on that partition.

PRESS any key on the keyboard to return to the Disk Backup and Maintenance Menu.

REPEAT the steps on this page to verify disks (partitions) F, G and H.

You're now ready to check out the system with the Diagnostic Exerciser disk. The instructions for running diagnostics start on page 8.

RUNNING DIAGNOSTICS

Before you run diagnostics, you'll need to put paper in your printer. Use two sheets so that the paper covers the width of the platen, and turn on the printer. The On/Off switch is on the front of the 620 printer (20 CPS) and on the back of the 630 printer (40 CPS).

As the test runs, you'll need to watch the screen for error messages. You may wish to read the procedures below to acquaint yourself with the messages before beginning the test.

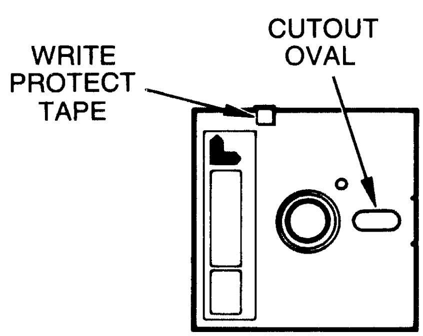
PRESS the RESET switch on the front of the 630 printer.

PRESS the RESET button on the back right corner of the screen or the CTRL+ESC keys on the keyboard.

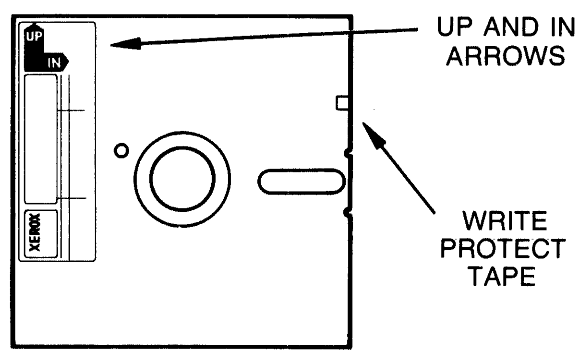
REMOVE any disks from the disk drives.

CHECK the Xerox 820-II Diagnostic Exerciser disk to be sure that it is not write protected. (The 5¼" disks are not protected when the tapes are removed, and the 8" disks are not protected when the tapes are on.)

INSERT the 820-II Diagnostic Exerciser disk in the left disk drive (drive A) and close the drive.



5¼" DISK



8" DISK

Note: The disk marked 820-II Diagnostic Exerciser disk must be used to run the diagnostic test. If you use the 820 Diagnostic Exerciser disk, a false error indication will occur.

IF you are testing a dual floppy disk drive system, insert an initialized disk in the right disk drive (drive **B**) and close the drive.

TYPE the letters **LA** and press the RETURN key.

The Xerox 820-II Diagnostic Exerciser disk will begin to check out the system. Be careful not to touch any keys on the keyboard while the test is running. Touching a key can stop the test.

Note: If there is an error during the test, holding down CTRL while pressing C will run the remaining tests.

WATCH the screen for the results of the first memory test. In about 30 seconds, the screen should show:

```
PASSES COMPLETE = 0001 ;  COUNT OF ERROR BYTES = 0000
PASSES COMPLETE = 0001 ;  COUNT OF ERROR BYTES = 0000
```

If the COUNT OF ERROR BYTES does not equal 0000, your system needs to be serviced.

WATCH the screen for the results of the second memory test. The screen should show:

```
PASSES COMPLETE = 0001 ;  COUNT OF ERROR BYTES = 0000
PASSES COMPLETE = 0001 ;  COUNT OF ERROR BYTES = 0000
```

If the COUNT OF ERROR BYTES does not equal 0000, your system needs to be serviced.

WATCH your screen for the results of the disk test. The disk drives will click during this test. When finished the screen should show:

```
0 read/write errors detected
0 seek errors detected
```

WATCH the screen test as it displays the screen test. The test pattern should fill the screen with characters. (The border around the test pattern will remain black.)

If there are missing characters or irregularities in the test pattern, your system needs to be serviced.

(If you do not have a printer, the test will end here. Remove both disks and press the RESET button.)

WAIT while the printer prints its test pattern. (If you have an 88 or 92 character wheel on the Printer, the test pattern will have blanks in some places.)

```
Normal print forward and back
 !"#$%&'()*+,-./0123456789:;¼=½?@ABCDEFGHIJKLMNOPQRSTUVWXYZ[•]•_°abcdefghijklmnopqrstuvwxyz§¶†™ !"#$%&'()*+,-./0123456789:;¼=½?@SUB
 !"#$%&'()*+,-./0123456789:;¼=½?@ABCDEFGHIJKLMNOPQRSTUVWXYZ[•]•‾°abcdefghijklmnopqrstuvwxyz§¶†™ !"#$%&'()*+,-./0123456789:;¼=½?@SUB

Interleaved overprinting, forward and backward
 !"#$%&'()*+,-./0123456789:;¼=½?@ABCDEFGHIJKLMNOPQRSTUVWXYZ[•]•_°abcdefghijklmnopqrstuvwxyz§¶†™ !"#$%&'()*+,-./0123456789:;¼=½?@SUB
 !"#$%&'()*+,-./0123456789:;¼=½?@ABCDEFGHIJKLMNOPQRSTUVWXYZ[•]•‾°abcdefghijklmnopqrstuvwxyz§¶†™ !"#$%&'()*+,-./0123456789:;¼=½?@SUB

Print same line once forward and backward
 !"#$%&'()*+,-./0123456789:;¼=½?@ABCDEFGHIJKLMNOPQRSTUVWXYZ[•]•_°abcdefghijklmnopqrstuvwxyz§¶†™ !"#$%&'()*+,-./0123456789:;¼=½?@SUB

Overprint with absolute horizontal tabbing
S!"#$%&'()*+,-./0123456789:;¼=½?@ABCDEFGHIJKLMNOPQRSTUVWXYZ[•]•_°abcdefghijklmnopqrstuvwxyz§¶†™ !"#$%&'()*+,-./0123456789:;¼=½?@SUB
```

If the test pattern did not print the alphabet and numbers, or if the characters were not properly aligned on the paper as shown in the example above, check the printer switches described in the SYSTEMS COMPONENT section on page 18 for the 620 Printer (20 CPS) or page 35 for the 630 CPS Printer (40 CPS). If the switches are correctly set, your system may need to be serviced.

WAIT for the following message to be displayed on the screen:

> "Diagnostics Complete - Touch ANY KEY to reset machine

REMOVE the Xerox 820-II Diagnostic Exerciser disk, the initialized disk, and place them back in their disk envelopes. Store your Diagnostic Exerciser disk in a safe place.

If all the test messages corresponded with those shown in the instructions, your system is hooked up correctly and in proper working order.

To learn how to use the basic features of CP/M on the 820-II, you should go through the CP/M Handbook. Then go through the manuals for any Applications software (such as Supercalc, TTY Communications, Accounting, etc.) to learn the specific features of that software.