ADOS

ADDS' DISK OPERATING SYSTEM

FOR

SYSTEM 50 AND SYSTEM 70

Software Release 1.4
Pub. # 512-31500D
November, 1978

11/78

This revision of the ADOS Reference Manual for System 50 and System 70 coincides with ADOS software release 1.4.

This manual is a prerequisite for ADDS*BASIC™ and ADDS* FORTRAN™.

Major portions of this manual were based on material supplied by Digital Research (CP/M).

A list of changed pages appears below with a margin bar on affected text pages to indicate the changes.

Page No.

iii	8-2
iv	8-3
v	8-4
1-2	8-5
2-3	8-6
2-4	8-7
3-2	8-8
3-3	
3-5	
3-6	
3-8	
3-9	
3-14	
3-15	
3-16	
3-20	
3-21	
3-22	
3-23 new	
4-2	
4-3	
4-4	
4-5	
4-6	
4-7	
4-8 new	
4-9	
4-10	
4-11	
4-12	
4-13	
4-14 new	
4-15 new	
4-16 new	
4-17 new	
4-18	
5-20	
5-21	
5-22	
5-23	

The material contained in this document is furnished for customer reference only, and is subject to change. The techniques described are proprietary and should be treated accordingly.

TABLE OF CONTENTS

TABLES

FIGURES

SECTION 1:

INTRODUCTION

ADOS, the ADDS Disk Operating System, was developed to provide a general environment for assembly-level language program construction, storage and editing. It also acts as an executive for the Assembler, Linking Loader, BASIC Interpreter, FORTRAN Compiler, and the Debug Utility (used by the Assembler, FORTRAN and BASIC modules).

ADOS 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.

Customers already familiar with System 70 using the ADDS*PLUS™ programming language will find that many of the unique features of System 70 have been incorporated into the packages described in this manual. Thus, the keyboard's special keys may be used to advantage, and files created on a System 70 under ADDS*PLUS™ may also be used by ADDS*FORTRAN™ and ADDS*BASIC™ programs on a System 70 or a System 50.

Other sections of this manual contain detailed procedures for using the Editor, Assembler, Linker, and Debug Utility.

Other manuals in this series include:

ADDS*FORTRAN™ Reference Manual
ADDS*BASIC™ Reference Manual

ADOS STRUCTURAL OVERVIEW

ADOS is divided into distinct parts. It contains a **Command Processor**, a disk operating system, an I/O subsystem and a temporary program area.

The **Command Processor** provides a symbolic interface between the System 50/70 keyboard as the console, and the remainder of the system. The **Command Processor** reads the keyboard and processes commands. These include listing the file directory, printing and displaying the contents of files, controlling the operation of the Assembler, Compiler, Interpreter, Linker, Editor and Debug Utility and initiating the Executive Processor (SUBMIT) for batch processing.

The disk operating system implements disk allocation strategies by controlling one or more disk drives containing independent file directories. These allocation schemes 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 diskette. In a standard system, each disk can contain up to 254 distinct files. A file can contain a maximum of 232 1K blocks, giving 232K of user area.

ADOS allows the user to look for a particular disk file by name, open and close files, change the name of a particular file, read and write a record from/to a file, and select a particular disk drive for further operations.

The I/O system provides the operations necessary to interface a line printer, floppy disk drive(s), CRT and keyboard.

The temporary program area holds programs which are loaded from the disk under control of the **Command Processor**. During program editing, for example, this area holds the Editor machine code and data areas. Similarly, programs created under ADOS can be checked out by loading and executing them in the temporary program area.

SECTION 2:

SYSTEM 50/70 FACILITIES

ADDS' System 50 and System 70 combine operational simplicity with the latest in technological sophistication and microprocessor capability. The heart of the system is an 8080 microprocessor with extensive addressing capabilities.

The CRT presentation is one of the largest available in the industry. It consists of 25 lines by 80 characters.

The keyboard is arranged in a standard typewriter layout with an additional 10-key numeric pad, complete cursor controls and a full set of named function controls for operational simplicity.

There are no unusual or unmarked keys, nor are double key depressions or switch sequencing normally required. An audible key click is standard, as is a keyboard security lock and an audible tone to alert the operator to special conditions.

The diskette storage unit is an IBM 3740-compatible floppy disk. Additional disk drives are available as an option and are useful for diskette copying and file storage.

A selection of printers is also available with speeds of from 30 characters per second to 340 lines per minute. Up to four stations can share one printer when the printer pooler interface option is attached.

System 50/70 contains its own diagnostics and self-test functions which are always initiated automatically at system start-up. System 70 consists of 12 printed circuit cards which are easily maintained on a board replacement basis. System 50 contains only nine printed circuit cards.

The system is housed in an attractive work station, designed to blend into an office environment. The CRT is mounted on the top surface of the desk. It was designed to reduce glare on the CRT screen from overhead lighting; thus available work space is increased and aesthetics are enhanced.

The keyboard and operator controls are mounted on the CRT directly in front of the operator. The diskette drive(s) are located in the pedestal of the desk at a convenient height for operator loading and unloading of diskettes.

Figure 2-1 shows the CRT screen.

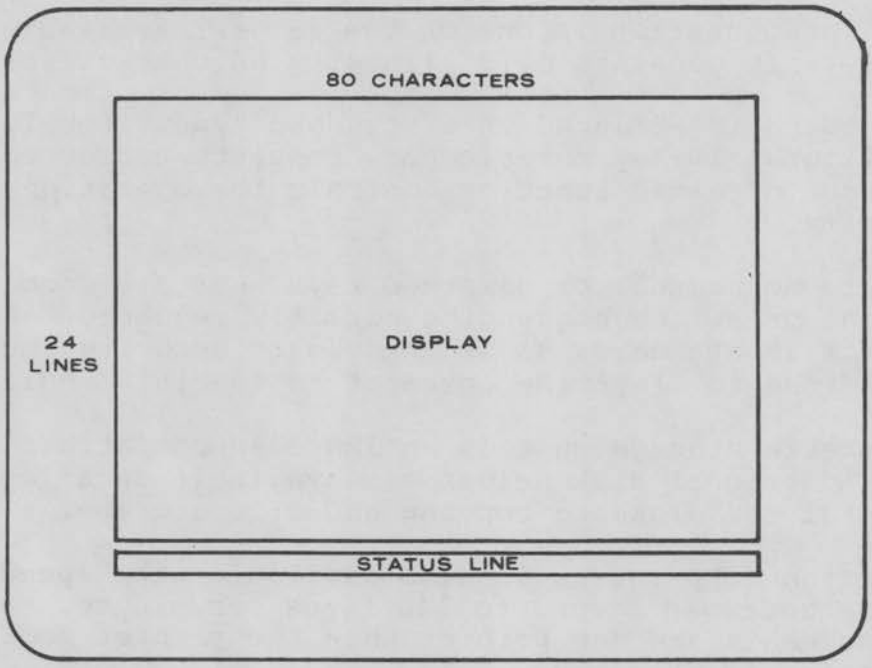


Figure 2-1 CRT SCREEN

Increased legibility is achieved by displaying characters as dark on a light background. A switch is available to reverse this pattern, if desired.

The character set includes all 128 ASCII characters, using a 7 X 12 dot space which accommodates descended lower case characters for easier reading. The cursor position is marked by an underscore symbol located under the character space.

The keyboard is described in detail in the System 70 Product Description. However, keys which have particular use and significance under ADOS are described here.

CTRL	The CTRL key is used in conjunction with other keys to perform special functions. In all cases, the CTRL key is held down while depressing the second key. Some of these functions are described here; other functions are described in appropriate sections of this manual.
CTRL-1	Depression of these keys during startup loads the disk operating system. Note that only the 1 (one) key from the numeric pad may be used for a program load.
CTRL-C	Depression of these keys performs the same function as the RESET key; the system is rebooted.
CTRL-P	Depression of these keys turns the attached line printer on and off. The printer is turned off automatically whenever the system is rebooted.
CTRL-S	Depression of these keys interrupts a listing on the screen. Depressing any other key causes the listing to continue. Depressing the RESET key, CTRL-C or any other key twice causes termination of the listing and returns control to the **Command Processor**.
CTRL-U	Depression of these keys deletes the current line. The cursor, which is positioned in column one of the next line, is not preceded by the prompt.
CTRL-Y	Returns control to the BASIC Interpreter after a double depression of the RESET key. The BASIC program previously loaded remains in the Temporary Program Area.
CTRL-Z	These keys act as a string terminator for the PIP parameters and the text editor, TED. When these keys are depressed only the symbol appears on the screen.
CURSOR CONTROLS	The HOME, left- and right-arrow keys allow free movement of the cursor within the logical line.
DEL CHAR	When this key is depressed, characters are deleted at the current cursor position.

ENTER	Each command to the **Command Processor** is executed only after the ENTER key is depressed.
ERASE	Depression of this key causes the balance of the current line from the cursor to be erased.
FWD TAB/ BACK TAB	Moves the cursor to the first non-space character of a word.
INS CHAR	When this key is depressed, it lights and data may be inserted at the current cursor position. To leave the insert mode, depress the INS CHAR key again. If editing is complete, simply depress the ENTER key.
LOCK/SHIFT	The operator depresses SHIFT to enter upper case letters (A-Z) or, when the LOCK key is down, shift to enter lower case letters.
PRINT LOCAL	When this key is depressed, the data currently on the screen is printed in its entirety on the attached printer.
RESET	Two successive depressions of this key aborts the current sequence and returns the operation to the **Command Processor** after rebooting the system (except in the Debug Utility, see Section 9). Important Note: After switching disks on any drive, the RESET key must be depressed twice to construct a new map of the directory.
SHIFT-ERASE	Depression of these keys erases the entire screen.

Additional uses for special keys are given in Section 4 (Editor).

SECTION 3:

COMMAND PROCESSOR

The user interacts with ADOS primarily through the **Command Processor** which reads and interprets commands entered at the System 50/70 keyboard. In general, the CP addresses one or more disk drives, labeled Drive A or B. A drive is considered to be logged in if the CP is currently addressing it. The currently logged drive is clearly indicated through the "prompt" which consists of the drive name (A or B) followed by the "greater than" symbol (>), indicating that the CP is ready and awaiting a command.

START-UP PROCEDURES

To load ADOS, ADDS' disk operating system, follow the steps below.

1. Turn the security lock to the OFF position.

2. Push the PRINT FORM Switch down to the OFF position.

3. Push the Power ON/OFF Switch up to the ON position.

4. The Status Line will contain the blinking message "TESTING." The system is in a mode in which all parts are automatically tested.

5. If testing is completed successfully, the Status Line will contain the message "LOAD PRGM."

6. If any problems are detected, another message will be displayed in the Status Line. (See Appendix A for a list of Test Mode Messages.)

7. To retest the system, depress the PREV PAGE key. Steps 4 and 5 will be repeated.

8. To bypass a printer buffer memory failure or communications processor error (System 70 only), depress the NEXT PAGE key.

9. Insert the diskette labeled ADOS-ADDS DISK OPERATING SYSTEM with the label side towards the open door latch handle, bottom first.

10. The diskette will click into place. If not, remove the diskette and repeat step 9.

11. Close the latch door handle (pull handle toward operator).

12. Hold down the CTRL key while depressing the 1 (one) key from the numeric pad. The red light on the latch button will be lit during the load operation.

After program load, the following message is displayed:

```
nnK ADOS VER m.m.
```

where

nn is the number of kilobytes of memory in the current system, and

m.m. is the version number.

Drive A is then automatically logged in, and the user is prompted with the symbol A> indicating that drive A is currently being addressed and commands may now be issued.

The commands are implemented at two levels--built-in commands and non-resident utility commands. Built-in commands are part of the CP itself, while non-resident utility commands are loaded into the temporary program area from disk and executed.

CHANGING THE LOGGED-IN DRIVE

The operator can change the logged-in drive simply by typing the disk drive name (A or B), followed by a colon (:) when the **Command Processor** is awaiting keyboard input. In the sequence shown below, the operator entries are underlined; the system responses are not.

```
nnK ADOS  REV 1.4

A>DIR *.FOR          List all the FOR files on the disk
                     in drive A.

A: INVT.FOR     PAYROLL.FOR  ACCT.FOR     TEST2.FOR

A>B:                 Log in drive B

B>DIR *.MAC          List all the MAC files on the disk
                     in drive B.

B: SALES.MAC    PGM1.MAC

B>A:                 Log in drive A

A>
```

SWITCHING DISKS

After switching disks on any drive, the RESET button must be depressed twice in succession to construct a new map of the directory. Failure to depress the RESET key (twice) will result in the diskette being identified as Read-Only (R/O) (see STAT command).

GENERAL COMMAND STRUCTURE

Both built-in commands and non-resident utility commands start with a command word and may be followed with other required data. In some cases, a reference to a particular file or files is required.

File References

A file reference can be specific or general. A specific file reference uniquely identifies a single file, while a general file reference may be satisfied by a number of different files.

File references consist of two parts--a primary name (filename) and a secondary name (filetype). Although the secondary name is optional, it is usually generic. For example, the secondary name MAC is used to denote an assembly language source file, while the primary name distinguishes each particular source file. The two names are separated by a period, as shown below.

pppppppp.sss

In this name, pppppppp represents the primary name of 8 characters (or less) and sss is the secondary name of no more than 3 characters. If only the primary name is used, the secondary name is assumed to consist of 3 blanks for all commands intrinsic to ADOS.

Table 3.1. Generic Filetypes

Generic Name	Filetype
BAK	Backup File--produced by Editor
BAS	BASIC Source File
COM	Command File--memory image (object) file
CRF	Cross Reference File
DAT	Data File
FOR	FORTRAN Source File
MAC	Assembler Source File
PRN	Assembler or Compiler Listing File
REL	FORTRAN or Assembler Relocatable Module
SUB	Submit File--used for batch processing
$$$	Temporary File--created by Editor and Utilities, then erased

The filename and filetype used to designate a specific file reference may not contain any of the following special characters:

. , ; : = ? *

A general file reference is used for directory search and pattern matching. The form of a general file reference is similar to a specific file reference, except that the question mark (?) may be interspersed throughout the primary and secondary names. In various commands, the question mark indicates that any file name constitutes a match if it matches exactly in all character positions except where the question mark appears. Thus, the general reference to a file name

X?Z.A?S

is satisfied by the specific file names

XYZ.AOS and
X3Z.ADS

The general file reference *.* is equivalent to ????????.??? while pppppppp.* and *.sss are abbreviations for pppppppp.??? and ????????.sss respectively.

The command DIR or DIR *.* is interpreted as a command to list the names of all disk files in the directory (on the screen), while DIR X.Y searches only for the file named X.Y.

The command DIR X?Y.A?S causes a search for all file names on the disk which satisfy this general file reference.

Command Conventions

The **Command Processor** always translates lower case characters to upper case characters. Lower case alphabetics are treated as if they are upper case in commands and filename references. In the command descriptions which follow, the following abbreviations and symbols are used:

drv:	denotes the disk drive to be used (usually A: or B:)
gfn	denotes a general file name
sfn	denotes a specific file name
<E>	indicates the ENTER key must be depressed to execute the command
CON:	indicates the screen
LST:	indicates the attached printer
[ ]	indicates optional items in command lines but must be used to enclose PIP parameters

(see page 3-13)

NOTE

Any time a general file reference is called for, a specific file reference may also be used.

Both the built-in commands and the non-resident utility commands execute from the logged-in drive (the drive named in the current prompt) unless the filename is preceded by a disk specifier (drv:).

1. Built-in Commands

a. ERASE Form: ERA [drv:]gfn <E>

The ERASE command removes files satisying the general reference gfn.

Examples:

A>ERA X.MAC <E>	The file named X.MAC is removed from the disk directory on drive A and the space is returned (can be used by new files).
B>ERA X.* <E>	All files with the primary name X are removed from the disk in the logged-in drive (drive B).
A>ERA B:*.DAT <E>	All files with the secondary name DAT are removed from the disk in the specified drive (drive B).

MESSAGES DISPLAYED:

FILE NOT FOUND

X.MAC ERASED

ALL FILES (Y/N) This message is displayed when the command

ERA *.*

is entered. A response of Y erases all files on the disk; a response of N aborts the command.

b. DIRECTORY Form: DIR [[drv:]gfn] <E>

The DIR command causes the names of all files which satisfy the general file reference gfn to be listed on the screen. If the general file reference is omitted, a listing of all files is provided.

Examples:

A>DIR <E> or A>DIR *.* <E>	All files on the disk in the logged-in drive (drive A) will be listed on the screen.
B>DIR *.COM <E>	All files on the disk in the logged-in drive (drive B) with the secondary name COM will be listed.
A>DIR B:X.DAT <E>	The specific file named X.DAT on the disk in the specified drive (drive B) will be listed.

MESSAGE DISPLAYED:

NOT FOUND

New Old

c. RENAME Form: REN [drv:]sfn=[drv:]sfn <E>

The REN command allows the user to change the names of files on disk. The file satisfying the old specific file reference is changed to the new specific file reference.

Examples:

A>REN X.MAC=Q.MAC <E>

The file previously named Q.MAC on the disk in drive A is changed to X.MAC.

A>REN B:NEW.DAT=B:OLD.DAT <E>

The file previously named OLD.DAT on the disk in drive B is changed to NEW.DAT.

MESSAGES DISPLAYED:

FILE NOT FOUND

OLD.DAT RENAMED TO NEW.DAT

FILE EXISTS

d. SAVE Form: SAVE n [drv:]sfn <E>

The SAVE command stores an exact image of n 256-byte blocks of the temporary program area beginning at 11ØØH. If the image saved is an executable file, the file name must have the filetype COM. At any time thereafter, the absolute machine code file can be loaded and executed simply by typing the filename without the filetype. If saving a new version under an old name, the old file is automatically erased before the new one is written.

The number of blocks of the temporary program area to be saved, n, is provided for FORTRAN and assembly programs by the Linking Loader (L8Ø). When assembling non-relocatable code, n is calculated by subtracting the starting address from the ending address and dividing by 1ØØH which equals 256 bytes.

Examples:

A>SAVE 3 X.COM <E>	Saves 3 blocks in the file named X.COM on the disk in drive A.
A>SAVE 40 B:Q.COM <E>	Saves 40 blocks in the file Q.COM on the disk in drive B.

e. TYPE Form: TYPE [drv:]sfn <E>

The TYPE command causes display on the screen of the contents of the ASCII source file named sfn. If the specified file is not an ASCII file, the results are unpredictable.

Examples:

A>TYPE X.BAS <E>	The file named X.BAS on the disk in drive A is displayed on the screen. NOTE: The BAS file must have been SAVEd in ASCII.
A>TYPE B:Z.MAC <E>	The file named Z.MAC on the disk in drive B is displayed on the screen.

CHECK IF CTL-S FREEZES & UNFRZ DISPLAY (IT DOES)

2. Non-resident Utility Commands

These commands are specified in the same manner as built-in commands, and additional commands can be easily defined by the user. They are loaded from the system disk and executed in the temporary program area.

a. STAT

Form: STAT [[drv:]gfn] <E>

The STAT command causes the display of the number of bytes of storage remaining on diskette and will indicate whether the diskette is Read/Write (R/W) or Read-Only (R/O). A diskette is identified as Read-Only (R/O) if it has been inserted in a drive, replacing another diskette, and the user has neglected to depress the RESET key (twice) to cause the system to read the directory of the newly inserted diskette (see Switching Disks, page 3-3). A double depression of the RESET key will transform the status from R/O to R/W.

Examples:

```
A>STAT                          Lists the status of the
A: R/W, SPACE: 132K             diskette in drive A.

A>STAT B:                       Lists the status of the
                                diskette in drive B.
BYTES REMAINING ON B: 90K

A>
```

The command STAT will display the status of the diskette in drive A until a STAT B: is executed. Each subsequent STAT command will display the status of both diskettes as shown below:

```
A>STAT                          Lists the status of the disk-
A: R/W, SPACE: 132K             ettes in both drive A and
B: R/W, SPACE: 90K              drive B.

A>STAT B:                       Lists the status of the disk-
                                ette in drive B.
BYTES REMAINING ON B: 90K

A>
```

If a file reference is included with the command, an alphabetical list of all files satisfying the general file reference is provided. The number of sectors (128 bytes per sector), number of bytes (in 1K blocks), number of extents and disk drive are included for each file. An extent is a directory entry for 16 logical blocks. Large files require more than one extent.

Check
STAT VAL:
STAT DEV:
STAT CON:=CRT

Examples:

```
A>STAT *.*
```

Lists the status of all files on drive A in alphabetical order.

```
RECS  BYTS  EX D:FILENAME.TYP
 165   21K   2 A:BASIC.COM
  30    4K   1 A:CREF80.COM
  42    6K   1 A:DEBUG.COM
  44    6K   1 A:FORMAT.COM
  54    7K   1 A:L80.COM
  75   10K   1 A:M80.COM
  56    7K   1 A:PIP.COM
 154   20K   2 A:SBASIC.COM
  24    3K   1 A:STAT.COM
  10    2K   1 A:SUBMIT.COM
  66    9K   1 A:TED.COM
  38    5K   1 A:VERIFY.COM
BYTES REMAINING ON A: 132K
```

```
A> STAT B:*.FOR
```

Lists the status of all FOR-suffixed files on drive B in alphabetical order.

```
RECS  BYTS  EX D:FILENAME.TYP
  14    2K   1 B: ACCT.FOR
  30    4K   1 B: INVT.FOR
 151   19K   2 B: PAYROL.FOR
 178   23K   2 B: TEST.FOR
BYTES REMAINING ON B: 90K
```

```
A>STAT
A: R/W, SPACE: 132K
B: R/W, SPACE: 90K

A>
```

Lists the status of the diskettes in both drive A and drive B (since drive B was accessed by a previous STAT command).

If the STAT program is not on the logged-in drive when the STAT command is entered, the system will respond with

```
STAT?
```

Assuming the STAT program is on drive A, and drive B is the logged-in drive, to list the number of bytes of storage remaining on drive B, enter

```
B>A:STAT
```

To list the remaining storage on drive A, enter

```
B>A:STAT A:
```

b. PIP Form: PIP <E>
PIP Command Line <E>

PIP is the Peripheral Interchange Program which implements the fundamental media conversion operations necessary to print, copy and combine disk files. When entered in either of the above forms, PIP is loaded into the temporary program area and executed.

In the first form, PIP reads Command Lines directly from the keyboard, prompted with the asterisk (*) until RESET or CTRL-C is depressed to abort or end the operation or the ENTER key is depressed with no command preceding it. Each successive Command Line causes some media conversion to take place according to the rules given below.

The second form is equivalent to the first, except that the single Command Line is automatically executed, and PIP terminates immediately with no further prompting for additional Command Lines.

The form of a PIP command is:

```
[drv:]dest=[drv:]file1,[drv:]file2,....,
           [drv:]filen <E>
```

check if file can be LOGICAL DEVICE A/O PHYSICAL DEVICE

where

dest is the file or peripheral device to receive the data -- the destination.

file1..n represents a file or a series of files which are copied (from left to right) to the specified destination. If more than one .BAS file is specified, the files must be in ASCII format--SAVEd with the A option (see SAVE statement in the ADDS*BASIC™ Reference Manual).

The equal sign (=) can be read as a left-pointing arrow.

destination <----- source

The destination and source elements can be specific file name references to source files, with or without a preceding drive name (A:, B:) which defines the particular drive to fetch or store the file. When the drive name is not included, the currently logged-in drive is assumed.

If the destination file already exists, it is replaced if the Command Line is properly formed; it is not replaced if any error condition arises.

Examples:

```
A>PIP <E>
```

Start PIP for a sequence of commands, expecting the asterisk (*) prompt.

```
*NEW.TEST=B:OLD.TST <E>
```

Move a copy of OLD.TST from drive B to the currently logged-in drive (drive A) and name the new file NEW.TEST. File OLD.TEST on drive B remains unchanged.

```
*B:X.FOR=Y.FOR <E>
```

Move a copy of Y.FOR on the currently logged-in drive to drive B and name the new file X.FOR.

```
*X.MAC=B:Y.MAC,B:Z.MAC,FIN.MAC <E>
```

Create the file X.MAC on the currently logged-in drive from the concatenation of the assembly language files Y and Z on drive B and FIN, also on the logged-in drive.

The destination 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 (see the example below).

```
*Y.FOR=X.FOR,Y.FOR,Z.FOR <E>
```

Replace the file Y.FOR with the concatenation of X.FOR, Y.FOR and Z.FOR.

The destination may also be the attached line printer or the screen (see the examples below).

```
A>PIP LST:=P.PRN <E>
```

Copy file P.PRN from drive A to the printer device (LST) and terminate the PIP program.

```
A>*CON:=X.MAC,Y.MAC,Z.MAC <E>
```

Concatenate the three .MAC files from the currently logged-in drive and copy to the screen (CON).

General filenames can be used as the source file if the disk drive name is stated as the destination alone. When a general file reference is used the PIP Utility will type (on the screen) all filenames as they are copied. See the examples below.

Examples:

```
A>PIP B:=*.*<E>
```

Copies all files from drive A, the currently logged-in drive, to drive B. The PIP Utility will type (on the screen) all filenames as they are copied.

```
B>PIP A:=*.COM<E>
```

Copies all .COM files from drive B to drive A.

```
COPYING -
M80.COM
L80.COM
STAT.COM
SYSGEN.COM
PIP.COM
TED.COM
DEBUG.COM
FORMAT.COM
SUBMIT.COM
F80.COM
VERIFY.COM
CREF80.COM

A>
```

The total Command Line length cannot exceed a physical line.

Lower case ASCII alphabetics are internally translated to upper case to be consistent with file and device name conventions.

An End-of-File Mark (CTRL-Z) is inserted as the last character if the destination is an ASCII file. All files except .COM files are treated as ASCII files.

PIP Parameters

The user can also specify one or more PIP parameters, enclosed in square brackets and optionally separated by one or more blanks. Each parameter affects the copy operation, and the bracketed list of parameters must immediately follow the source file or device. Some parameters may be followed by an optional decimal integer value. The valid PIP parameters are listed below.

Dn	Delete the characters which extend past column n in the transfer of data to the destination from the character source. This parameter is useful in truncating long lines (such as extended comments in a program listing) which are sent to a narrow printer or screen.
E	Echo (display) all transfer operations on the screen as they are being performed.
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 one.
O	Object file (non-ASCII) transfer. The normal end of file, 1AH, is ignored.
Qstring	Quit or stop copying from the source device or file when the (required) string is encountered. This is terminated by a CTRL-Z.
Sstring	Start copying from the source device when the (required) string is encountered. This parameter is also terminated by CTRL-Z. The S and Q parameters can be used to "abstract" a particular section of a file (such as a subroutine).
Tn	Expand the tabs (CTRL-I characters) to every nth column during the transfer of characters to the destination from the source.
U	Translate the lower case alphabetics to upper case during the copy operation.
V	Verify that data has been copied correctly by re-reading after the write operation. The destination must be a disk file.

Examples of PIP command lines containing the parameters described above:

```
A>PIP X.COM=B:[VO]  <E>
```

Copy the non-ASCII file X.COM from drive B to the current drive (drive A) and verify that the data was properly copied. When the name of the source file is omitted, it is assumed to be the same as the name of the destination file.

```
B>PIP LST:=X.MAC[NT8U]  <E>
```

Copy file X.MAC from drive B to the printer; number each line (N), expand tabs to every 8th column (T8), and translate lower case alphabetics to upper case (U).

```
A>PIP X.LIB=Y.MAC[SSUBR1: QJMP^L3^] <E>
```

Copy part of the file Y.MAC into the file X.LIB. Start the copy (S) when the string "SUBR1:" has been found and quit copying (Q) after the string "JMP L3" is encountered.

```
*X.MAC=Y.MAC[V],Z.MAC[N]  <E>
```

Create the file X.MAC on the currently logged disk from the concatenation of the .MAC files Y and Z. Verify (V) file Y.MAC and number (N) each line of Z.MAC.

CAUTION

The PIP Utility cannot be used to copy diskette files created under pre 1.4 versions of ADOS to an ADOS 1.4 diskette. Also, the FORTRAN library (FORLIB.REL), the Compiler (F80) and the Linking Loader (L80) are unique for each version of ADOS, so care should be taken when using the PIP program to assure that the source and destination diskettes contain the same version of ADOS.

c. SYSGEN Form: SYSGEN <ENTER>

This command allows generation of an initialized diskette containing ADOS. The SYSGEN program prompts the operator with interaction as shown.

SYSGEN <ENTER> Initiates the SYSGEN.

*SYSGEN VERSION 1.4 SYSGEN sign-on message.

SOURCE DRIVE NAME (OR DEPRESS <E>)

The user may enter only A or B. When A or B is entered, the system uses the entry to read the master file from the specified drive (drive A or B). (Depression of the ENTER key is not implemented at present.)

SOURCE ON [A] [B] THEN DEPRESS <E>

Place a diskette containing ADOS on the specified drive and depress ENTER when ready.

FUNCTION COMPLETE System has been copied to memory.

DESTINATION DRIVE NAME (OR <E> TO REBOOT)

The user may enter A or B or depress the ENTER key. If either A or B is entered, the system uses the entry as the destination drive (drive A or B). If the ENTER key is depressed, the SYSGEN is aborted and the system reboots.

DESTINATION ON [A] [B] NOW DEPRESS <E>

Place a diskette into the specified drive and depress the ENTER key when ready.

FUNCTION COMPLETE The diskette is initialized on the specified drive.

DESTINATION DRIVE NAME (OR <E> TO REBOOT)

The user may enter A or B or depress the ENTER key. If either A or B is entered, the system will generate another initialized diskette using the entry as the destination drive (drive A or B). If the ENTER key is depressed, the SYSGEN Utility is exited and the system reboots.

Upon completion of a successful system generation on a new diskette, the diskette will contain only ADOS and resident commands. If other programs are already on the disk, they will remain untouched and available.

A factory-fresh IBM-compatible diskette appears to ADOS as a diskette with an empty directory. The operator may copy appropriate files from an existing ADOS diskette (having the same version ADOS operating system) to the newly constructed diskette using the PIP Utility. The command

```
A>PIP B:=*.*
```

will copy all files from drive A to drive B.

Note 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 drive B and will never be the source of a bootstrap operation on drive A, the SYSGEN need not take place and in fact a new diskette needs absolutely no initialization to be used with ADOS.

If data diskettes formerly created by ADDS*PLUS™ are to be re-used (media only) under ADOS, it is necessary that they first be erased using the FORMAT program (see page 3-20).

However, ADDS*PLUS™ data files which are to be either read or re-written need no special initialization.

NOTE

Both ADOS and ADDS*PLUS™ files cannot appear on the same diskette.

d. SUBMIT Form: SUBMIT [drv:]sfn paraml,...n<E>

The SUBMIT command allows ADOS commands to be grouped together for automatic batch processing. The sfn given in the command must be the name of an existing file with an assumed filetype of SUB. The .SUB file contains prototype commands with possible parameter substitution. The actual parameters (paraml,...n) are substituted into the prototype commands and, if no errors occur, the file of substituted commands is processed sequentially.

The command file is created using the Editor with interspersed dollar sign ($) parameters of the form $1, $2, $n, corresponding to the actual number of parameters which will be included when the file is submitted for execution. The maximum number of parameters which may be given is 10. When the SUBMIT command is executed, the actual parameters are paired with the formal parameters in the prototype commands (paraml for $1, param2 for $2, etc.). If the number of formal and actual parameters does not agree, the SUBMIT function is aborted and an error message is displayed on the screen.

The SUBMIT function creates a temporary file of substituted commands with the name $$$.SUB on the logged-in drive. When the system reboots at the termination of SUBMIT, this command file is read as a source of input rather than the keyboard. This feature is useful for next day automatic start-up procedures.

If the SUBMIT function is performed and the logged-in drive is not drive A, the commands will not be processed.

The user can abort command processing at any time by depressing the RESET key. In this case, the $$$.SUB file is removed, and the subsequent commands come from the keyboard. Command processing is also aborted if an error is detected in any of the commands.

Example of contents of file ASMBL.SUB:

```
M80 $1,$1.$2=$1
DIR $1.*
ERA $1.BAK
PIP LST:=$1.$2
ERA $1.$2
```

The operator issues the command

```
SUBMIT ASMBL X PRN <E>
```

and the SUBMIT program reads the ASMBL.SUB file. X is substituted for all occurrences of $1 and PRN is substituted for all occurrences of $2, resulting in a $$$.SUB file as shown below.

```
M80 X,X.PRN=X
DIR X.*
ERA X.BAK
PIP LST:=X.PRN
ERA X.PRN
```

These are executed in sequence.

If a SUBMIT command is the last command in a .SUB file, it initiates another .SUB file, thus providing for chained batch commands.

e. VERIFY Form: VERIFY <E>

The VERIFY program provides a simple method of verifying the integrity of a diskette prior to committing valuable data to the diskette. To invoke the VERIFY Utility, type

VERIFY <ENTER>

The utility responds with

PLACE DISK TO BE VERIFIED IN DRIVE B:
TYPE "C <ENTER>" TO CONTINUE

After inserting the diskette to be verified in drive B, depress

C <ENTER>

If the response is incorrect, the utility displays the following message

WRONG RESPONSE, TRY AGAIN

Depress C followed by the ENTER key.

The utility proceeds to verify the integrity of the diskette by reading every sector of every track. If all sectors are successfully read, the following message appears on the screen.

DISKETTE VERIFIED

If the utility fails to read any sector, the following message appears:

DISK B: BAD

In either case, the system reboots and the ADOS prompt A> appears on the screen.

f. FORMAT Form: FORMAT [gfn] <E>

The FORMAT program is a multi-purpose utility which combines the functions of the VERIFY Utility and the SYSGEN, PIP and SUBMIT commands. This utility

1) verifies the integrity of the diskette to be formatted,
2) initializes the directory of the unformatted diskette,
3) copies the ADOS resident system from an existing system diskette, and
4) optionally, copies one or more files from the existing system diskette.

To invoke the FORMAT Utility, enter one of the following commands

```
FORMAT  <E>
FORMAT S  <E>
FORMAT gfn  <E>
FORMAT gfn S  <E>
```

where gfn represents a general or specific file reference and the S option prevents the destruction of the files on the diskette to be formatted by inhibiting the initialization of the directory.

If a file reference is included, the FORMAT Utility copies all files satisfying the file reference from the diskette in drive A to the newly formatted diskette in drive B.

The command FORMAT S may be used in place of the SYSGEN command to initialize a diskette containing files created only under ADOS release 1.4. If additional files are to be copied to the destination diskette, the command FORMAT gfn S is recommended. However, if a file on the destination diskette has the same name as a file on the source diskette it will be overwritten by the file from the source diskette.

The utility responds with

```
PLACE DISK TO BE FORMATTED IN DRIVE B:
TYPE "C <ENTER>" TO CONTINUE
```

Unless directory initialization is suppressed, using the S option, the following warning will also be displayed:

```
***ALL DATA ON DISK B:  WILL BE DESTROYED***
```

Insert the diskette to be formatted in drive B. A previously used diskette can be formatted by this utility. Type C and depress the ENTER key.

If the response is incorrect, the utility displays the following message

WRONG RESPONSE, TRY AGAIN

Depress C followed by the ENTER key.

When the Format Utility has completed all its tasks, the following message appears on the screen:

FORMATTING COMPLETE

The system reboots and the ADOS prompt (A>) appears on the screen. If no file reference was specified, the operator may now resume entering commands.

If a file reference was specified, a PIP command line is set up including this file reference. This command is written out to a $$$.SUB file created by the FORMAT Utility on drive A before the system reboots. Then the PIP command

PIP B:=A:gfn[OV]

appears on the screen as the .SUB file is executed. The .SUB file is deleted from diskette A and the system reboots again.

NOTE

If the file reference specified is *.* the $$$.SUB file will be copied from drive A to drive B by the PIP command. This file must be manually deleted from the diskette in drive B. If not deleted, this file would be executed immediately when the formatted diskette is used to load the system or when a SUBMIT command is used.

If an error is encountered, the appropriate error message is displayed and the system reboots. All errors should be considered fatal. A table of possible error messages appears on the following page.

Table 3.2. Format Utility Error Messages

Message	Cause
CLOSE ERROR DISK A: SUBMIT FILE	An error was encountered while closing the $$$.SUB file, which now contains the PIP command.
CREATE ERROR DISK A: SUBMIT FILE	An error was encountered while creating a $$$.SUB file on diskette A: to hold the PIP command.
DISK B: BAD	An error was encountered while reading diskette B: to verify integrity.
HOME ERROR DISK x:	Diskette drive x: failed to home the head properly.
READ ERROR DISK A:	A read error was encountered while copying the system tracks (Ø,1).
WRITE ERROR DISK A: SUBMIT FILE	A write error was encountered while writing the PIP command out to the $$$.SUB file.
WRITE ERROR DISK B:	A write error was encountered while initializing the directory of diskette B: or while copying the system tracks (Ø,1).

CHANGING A TYPED COMMAND

Before the ENTER key is depressed, a typed command may be modified or deleted.

The following keys may be used to modify a typed command while the cursor is in the current line.

-> <-, HOME, DEL CHAR, INS CHAR, ERASE

These keys are described in detail in Section 2.

Depressing CTRL-U will delete the current line. Another command line may then be entered.

Additional ADOS Error Messages

Error Message	Description
SELECT PRINTER	An attempt has been made to access the printer and it is not selected.
PUT DISK IN n:	An attempt has been made to access a drive containing no diskette.
LOAD ADOS	An attempt has been made to load the ADOS program without a resident program load diskette.

SECTION 4:

EDITOR

TED--TEXT ORIENTED EDITOR

This Editor is similar to TECO (PDP-10) and the INTEL MDS Editor. It is invoked by typing TED when a prompt is on the screen indicating that the **COMMAND PROCESSOR** is awaiting a command, as shown:

A>TED [drv:]filename[.filetype][ødrv:] <ENTER>

If [drv] is omitted, it is assumed to be the logged-in drive (drive A) for both reading and writing. If the first [drv] is specified, and the second is omitted, the edited file will be written to the same diskette as the source file. If the filetype is omitted the system defaults to three blanks. The user may specify a different drive at the end of the command to cause the edited file to be written to the diskette residing on that drive.

TED reads segments of the source file (given by filename or filename.filetype) into memory under operator control. The source file is manipulated by the operator and subsequently written back to diskette after being altered. Upon exit from TED the name of the original file is changed from fn.ft to fn.BAK so that the most recent previously edited source file can be reclaimed, if necessary. A temporary file which is used during editing and called fn.$$$ is renamed to fn.ft, which then becomes the resulting edited file.

If the source file named does not already exist, it is created by TED and initialized to empty. The response "NEW FILE" appears on the screen. The user can then give the commands described in this section to enter data into the new file.

TED Operation

TED operates on the source file, shown in Figure 4-1 as fn.ft (filename.filetype), and passes all text through a memory buffer where the text can be viewed or altered. The number of lines which can be kept in the memory buffer varies with the line length, but the total capacity for each system size is given below.

Number of Characters	System Size
19,700	32K
23,900	40K
32,200	48K

Text material which has been edited is written onto a temporary work file called fn.$$$ 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.

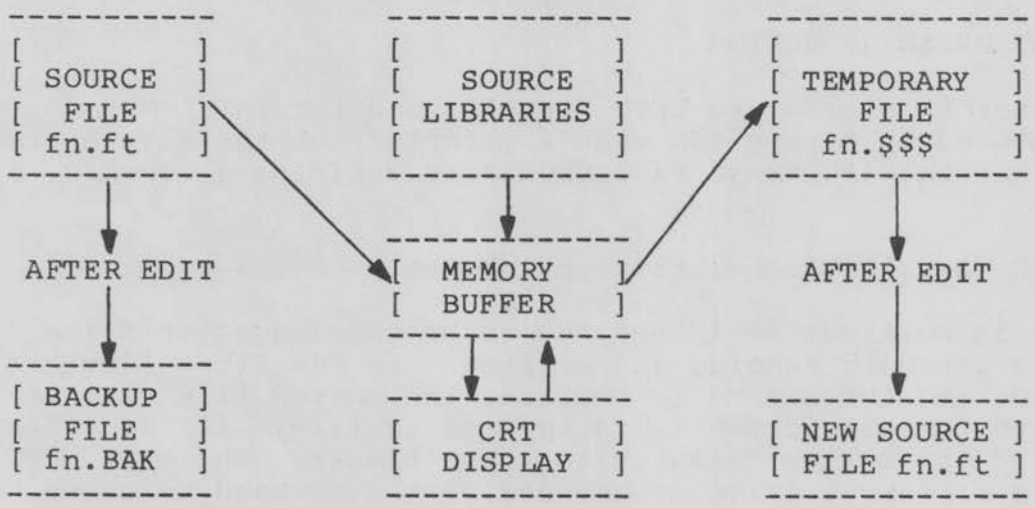


Figure 4-1. TED Operation

Upon initiation of TED, the memory buffer is empty. The operator may either bring in lines from a source file (using the append command A) or insert lines directly from the keyboard (using the insert command I). All commands are executed only after depression of the ENTER key.

An imaginary "character pointer" moves throughout the memory buffer under command of the operator. In the examples and descriptions to follow, the character pointer is referred to as CP. It is always located before the first character of the first line, after the last character of the last line, or between two characters. The current line is the source line which contains the CP.

Various commands can be issued which manipulate the CP or display source text in the vicinity of the CP. The commands are read directly from the keyboard prompted by the asterisk (*). The commands described with a preceding n indicate that an optional integer value can be specified. The integer n indicates either the number of times a command is to be executed or the number of lines or characters to be affected by the command.

The integer n can be unsigned or have an optional preceding minus sign. If no sign is given, then plus is assumed. If an integer n is optional, but is not supplied, n is assumed to be 1. The pound sign (#) is replaced by the integer 65535. In addition, the user can precede certain commands by absolute line numbers in the range 1 to 65535 (see page 4-8).

Any number of commands can be entered contiguously, up to the capacity of the buffer (128 characters). They are executed as a group after the ENTER key is depressed.

Table 4.1. TED Commands

COMMAND	DESCRIPTION
nA	Append the next n unprocessed source lines from the source file to the end of the memory buffer.
#A	Append up to 65535 lines of a file to the memory buffer.
ØA	Append source characters to the memory buffer until it is at least half full.
B or -B	Move the CP to the beginning of the memory buffer if unsigned and to the end if -.
nC or -nC	Move the CP by n or - n characters (forward in the buffer, if unsigned and back if -).
nD or -nD	Delete n characters ahead of the CP if unsigned, or behind the CP if -.
E	End the edit. Copy all buffered text to the temporary file. Copy all unprocessed source lines to the temporary file. Rename the files as shown in Figure 4-1.
nFstring	Find (locate) "string" starting at the current position of CP. The match is attempted n times, and if successful, CP is moved directly after the last character in "string." If the n matches are not successful, CP is moved directly after the last character in the last "string" located and an error message is displayed on the screen. The F command searches only the memory buffer. The number of characters in "string" is limited to 100.
H	Move to the head of the new file by performing an automatic E command. The temporary file becomes the new source file. The memory buffer is emptied. A new temporary file is created. This is equivalent to issuing an E command followed by a reinvocation of TED using fn.ft as the file to be edited.
I	I followed by the ENTER key causes TED to enter the Insert Mode. The TAB, HOME, left- and right-arrow keys and the INS CHAR and DEL CHAR keys are effective in the Insert Mode. After the necessary characters have been entered, depress CTRL-Z to exit the Insert Mode. The CP is positioned after the last character entered.

Table 4.1. TED Commands (Continued)

COMMAND	DESCRIPTION
	I followed by one line of data and the ENTER key causes TED to enter the Input Mode. In the Input Mode a single line of data will be entered before the CP. If the character string is terminated by CTRL-Z, the characters are inserted directly following the CP and the CP is positioned after the last character entered. In the Input Mode, the HOME, left- and right-arrow keys and the INS CHAR and DEL CHAR keys are effective.
nJstr1^str2^str3	Search from the current CP for the next occurrence of "str1." If found, insert "str2" after "str1" and move CP to follow "str2." Delete all characters following CP up to but not including "str3" leaving CP directly after "str2." If "str3" cannot be found, then no deletion is performed. The search continues until "str1" has been found n times or the end of the buffer has been reached. The number of characters in the three strings is limited to 100.
nK or -nK	Kill (remove) n or - 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 unsigned, while the characters after CP remain if - is specified.
nL or -nL	n=Ø move CP to the beginning of the current line if it is not already there. n<>Ø 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 unsigned) or n lines up (if -). The CP will stop at the top or bottom of the memory buffer if the value specified for n is too large.
nMstring	This is a MACRO command and is used to batch a "string" of TED commands together (not including another M command) for repeated evaluation. It is executed as follows: n=Ø, n=1 The command "string" is executed repetitively until an error condition is encountered (e.g., end of memory buffer is reached with an F command). n>1 The command "string" is executed n times.

Table 4.1. TED Commands (Continued)

COMMAND	DESCRIPTION
n or -n	This is equivalent to nLT or -nLT, which moves the CP up or down n lines and causes a single line to be typed. Depressing only the ENTER key causes the next line to be typed.
nNstring	Search the entire source file for the nth occurrence of "string." The operation of the N command is precisely the same as F above, except when n instances of "string" cannot be found within the current memory buffer. In this case, the entire memory contents are written with an automatic #W and input lines are then read until the memory buffer is at least half full, or the entire source file is exhausted. The search continues in this manner until "string" has been found n times, or until the source file has been completely transferred to the temporary file. The number of characters in "string" is limited to 100.
O	Return to the original file. The memory buffer is emptied. The temporary file, fn.$$$, is deleted. The effects of the previous editing commands following the last H are thus nullified. System response is (Y/N). Type Y to return to the original file.
nP or -nP	Move the CP n or - n pages (24 lines to a page) and type (display on the screen) the current page. The CP remains on the first line of this page.
Q	Quit (leave) the edit with no file alterations, returning immediately to the **COMMAND PROCESSOR**. The temporary file, fn.$$$, is deleted. The effects of the previous editing commands following the last H are thus nullified. System response is (Y/N). To quit, type Y.
R[filename]	The "library read" command reads the specified source file, placing the characters in the memory buffer after CP in a manner similar to the I command. Filenames in this command are assumed to have a filetype of LIB. If the filename is omitted, the default is the temporary file X$$$$$$$$.LIB (see X command). The R command does not empty the LIB file. For example, if the command RMYFIL is issued, TED reads from the file MYFIL.LIB until the end of file, and automatically inserts the characters into the memory buffer.

Table 4.1 TED Commands (Continued)

COMMAND	DESCRIPTION
nSstring1^string2	Search the memory buffer for "string1" starting at the current position of CP and successively substitute "string2" for "string1" until the end of the buffer, or until the substitution has been performed n times. This is equivalent to combining the F, D and I commands into one Command String. The number of characters in the two strings is limited to 100.
nT or -nT	n=Ø Type (display on the screen) the contents of the current line up to CP. n=1 Type the contents of the current line from CP to the end of the line. n<>1 Type the current line along with n-1 lines which follow, if unsigned; if -, type the previous n lines, up to the CP. To abort long typeouts without leaving TED, depress any key once.
U or -U	The U command translates lower case alphabetics to upper case. The command characters are echoed as typed, without translation. The -U command causes TED to revert to "no translation" mode. When TED is invoked, the default is -U (no translation).
V or -V	The V or Verify Line Numbers command prefixes absolute line numbers to each line in the buffer. The line number is displayed on the screen before each line in the following format: nnnnn: program line 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 five blanks. Absolute line numbers are produced only during the editing process and are not recorded with the file. These numbers change following deletion (K) or insertion (I) of lines of text. The -V command discontinues the display of the absolute line numbers. When TED is invoked, the default is -V.

Table 4.1 TED Commands (Continued)

COMMAND	DESCRIPTION
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) total — is the size of the memory buffer
nW	Write the first n lines of the memory buffer to the temporary file free space and shift the remaining lines to the beginning of the buffer.
#W	Write the entire memory buffer to the temporary file free space.
ØW	Writes lines to the temporary file free space until the buffer is at least half empty.
nX	Block move facility which transfers the next n lines including the current line to the temporary file X$$$$$$$.LIB which is active only during the editing process. Each execution of the nX command causes additional lines to be transferred to the temporary file, one set behind the other. They can be retrieved by typing R (see R command). Note that the X command does not remove the transferred lines from the memory buffer. Use the K (kill) command to delete these lines. In order to effect a "block move" of lines in an ADDS*BASIC™ program, the BASIC line numbers must be changed before invoking the BASIC Interpreter. If not, they revert back to their original locations as soon as the BASIC program is loaded.
ØX	This command empties the temporary LIB file. However, if an editing session is ended with either the Q or E command, this temporary file is automatically removed. If TED is aborted by depressing CTRL-C, the LIB file will exist if an X command had been given. A subsequent invocation of TED will erase this file.
nZ	Temporary pause (sleep) in the editing session for n seconds before the TED prompt (*) is returned.

Command Strings

Any number of commands up to the capacity of the buffer (128 characters) can be typed contiguously. These commands are executed only after the ENTER key is depressed (see A Sample Editing Session, page 4-11).

However, the following commands, which are considered potentially disastrous, must be entered separately rather than in command strings.

E — end the edit; deletes the backup file created by the previous editing session by renaming the current source file from fn.ft, to fn.BAK.

H — move to the head of a new file (by performing an automatic E followed by reinvocation of TED); also deletes the backup file created by the previous editing session as with the E command above.

O — return to the original file (by performing an automatic Q followed by a reinvocation of TED); deletes the temporary file, fn.$$$, containing the effects of the editing commands issued during the editing session (after the last H command).

Q — quit (leave) the edit; deletes the temporary file, fn.$$$, containing the effects of the editing commands issued during the editing session (after the last H command) and returns to the **Command Processor**.

Absolute Line Numbers

The user may reference an absolute line number by preceding the following commands by a number in the range 1 to 65535 followed by a colon:

nA	nC	nD	nF	nJ
nK	nL	nM	nN	nP
nT	nW	nX	nZ	

Command	Effect
35:T	Move the CP to absolute line number 35 and type the line.
80:4D	Move the CP to absolute line number 80 and delete the next four characters.
145:10W	Move the CP to absolute line number 145 and write the first 10 lines of the memory buffer to the temporary file free space.

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.

Command	Effect
:210T	Type from the current line number through the line whose absolute number is 210.
:23K	Delete lines from the CP backward or forward through absolute line number 23.

The above two absolute line reference forms may be combined as follows:

Command	Effect
35::210T	Move to absolute line 35, then type through line 210.
200::310A	Append lines 200 through 310 inclusive from the source file to the end of the memory buffer.
17::22K	Delete lines 17 through 22 inclusive from the memory buffer.

Error Messages

When an error occurs, TED prints an error message in the form:

BREAK "x" AT c

where

x is one of the error indicators listed below

c is the command where the error occurred.

Error Indicator	Meaning
?	Unrecognized command
#	Cannot apply the command the number of times specified (e.g., in an F command).
>	Memory buffer full; F, N or S strings too long. Use one of the commands D, K, N, S or W to remove characters.
O	Cannot open the LIB file specified in the R command.

CRC (Cyclic Redundancy Check) information is written with each output record in order to detect errors on subsequent read operations. If a CRC error is detected, the message "PERM ERR DISK D" is displayed where D is the currently selected drive (A or B). 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 insure that it contains the proper information, then removing the primary file and renaming the BAK file. The file can then be re-edited, starting with the previous version.

Table 4.2 Control Characters and Commands

CHARACTER	FUNCTION
CTRL-E	Physical CR/LF, not actually entered in command, used if the command line is too long
CTRL-I	Logical tab, same as the TAB key
CTRL-L	Logical CR/LF in search and substitute strings
CTRL-Z	String terminator

A Sample Editing Session

Write a FORTRAN program to calculate monthly mortgage payments and the total of these payments over the term of the loan.

1. Invoke the text-oriented editor, TED.

```
A>TED AMORT.FOR              <ENTER>    ;operator entry

NEW FILE                                ;system response
*_
```

2. Enter the Insert Mode and key in the FORTRAN statements, as shown below. Use the TAB key to move the cursor to the statement field. Depress the ENTER key after each statement has been entered.

```
*I                           <ENTER>


      PROGRAM AMORT
C     CALCULATES MONTHLY MORTGAGE PAYMENTS AND
C     TOTAL PAYMENTS OVER TERM OF LOAN
      WRITE (3,10)
10    FORMAT (2X,'ENTER PRINCIPAL')
      READ (3,15) P
15    FORMAT (F7.0)
      WRITE (3,20)
20    FORMAT (2X,'ENTER ANNUAL INTEREST RATE')
      READ (3,25) R
25    FORMAT (F7.0)
      WRITE (3,30)
30    FORMAT (2X,'ENTER TERM IN YEARS')
      READ (3,35) N
35    FORMAT (I3)
      MR=R/12
      PAY=(P*MR)/(1-(1/(1+MR)**(12*N)))
      TOT=PAY*12*N
      WRITE (3,40)
40    FORMAT (30X,'SUMMARY OF MORTGAGE PAYMENTS'//)
      WRITE (3,45)
45    FORMAT (2X,'PRINCIPAL',5X,'INTEREST RATE (%)',5X,
     X  '# OF YEARS',5X,'MONTHLY PAYMENT',5X,'TOTAL PAYMENT'/)
      WRITE (3,50) P,R,N,PAY,TOT
50    FORMAT (2X,F9.2,10X,F7.2,13X,I3,2(11X,F9.2))
      STOP
      END

_
```

3. Exit the Insert Mode by depressing CTRL-Z.

```
*_
```

5. Move the character pointer back to the beginning of the memory buffer and print five lines.

```
*B5T                        <ENTER>

        PROGRAM AMORT
C       CALCULATES MONTHLY MORTGAGE PAYMENTS AND
C       TOTAL PAYMENTS OVER TERM OF LOAN
        WRITE (3,10)
10      FORMAT (2X,'ENTER PRINCIPAL')
*_
```

5. Move the character pointer down three lines and print the next line.

```
*3LT                        <ENTER>
        WRITE (3,10)
*_
```

6. Insert the type statement REAL MR before the line just printed. The TAB key does not function in the Insert Mode.

```
*I          REAL MR         <ENTER>
        REAL MR*_
```

7. Move the character pointer back two lines and print the next four lines.

```
        REAL MR*-2L4T   <ENTER>
C       TOTAL PAYMENTS OVER TERM OF LOAN
        REAL MR
        WRITE (3,10)
10      FORMAT (2X,'ENTER PRINCIPAL')
*_
```

8. Change the specification in FORMAT statement 25 from F7.0 to F7.2 by "finding" the second occurrence of "25" (2F25) and using the "search and substitute" command S to effect the change. CTRL-Z is used as a string terminator for the F and S commands. When these keys are depressed, only the symbol ^ appears on the screen. Print the edited lines.

```
*2F25^S7.0^7.2^0LT      <ENTER>
25      FORMAT (F7.2)
*_
```

9. End the editing session.

```
*E                          <ENTER>

A>
```

10. Reinvoke TED and modify the above program so that the output is printed on the printer rather than the screen by changing the logical unit number (LUN) in the last three WRITE statements.

```
A>TED AMORT.FOR          <ENTER>
*_
```

11. Append source characters to the memory buffer until it is at least half full.

```
*0A                      <ENTER>
*_
```

12. Move the character pointer somewhere between statement 30 and the next WRITE statement. One way to accomplish this is to "find" the first occurrence of "35". This positions the character pointer between the "5" and the ")" of the statement READ (3,35) N.

```
*F35                     <ENTER>
*_
```

13. Using the MACRO command (M) batch commands to search for the next three occurrences of LUN 3, substitute LUN 2 and print the edited WRITE statement.

```
*3MS(3,^(2,^0LT          <ENTER>
       WRITE (2,40)
       WRITE (2,45)
       WRITE (2,50) P,R,N,PAY,TOT

BREAK "#" AT T
*_
```

14. Use the H command to save this version of the program as AMORT.FOR without exiting TED. Rename AMORT.BAK as AMYR.FOR at the end of the session. Note that the A command must be used following the H command to append source characters to the memory buffer.

```
*H                       <ENTER>
*0A                      <ENTER>
*_
```

15. Modify the program to output the monthly mortgage payments, total payments and the saving if the term of the loan is decreased by 2 years. The final version of the program appears on the next page.

FINAL VERSION OF AMORT.FOR

```
      PROGRAM AMORT
C     CALCULATES MONTHLY MORTGAGE PAYMENTS AND
C     TOTAL PAYMENTS OVER TERM OF LOAN
      REAL MR
      WRITE (3,10)
10    FORMAT (2X,'ENTER PRINCIPAL')
      READ (3,15) P
15    FORMAT (F7.0)
      WRITE (3,20)
20    FORMAT (2X,'ENTER ANNUAL INTEREST RATE')
      READ (3,25) R
25    FORMAT (F7.2)
      WRITE (3,30)
30    FORMAT (2X,'ENTER TERM IN YEARS')
      READ (3,35) N
      M=N
35    FORMAT (I3)
      WRITE (2,40)
40    FORMAT (30X,'SUMMARY OF MORTGAGE PAYMENTS'//)
      WRITE (2,45)
45    FORMAT (2X,'PRINCIPAL',5X,'INTEREST RATE (%)',5X,
     X  '# OF YEARS',5X,'MONTHLY PAYMENT',5X,'TOTAL PAYMENT'/)
      MR=R/12
55    PAY=(P*MR)/(1-(1/(1+MR)**(12*N)))
      TOT=PAY*12*N
      WRITE (2,50) P,R,N,PAY,TOT
50    FORMAT (2X,F9.2,10X,F7.2,13X,I3,2(11X,F9.2))
      IF (N.NE.M) GO TO 70
      TOT1=TOT
      N=N-2
      GO TO 55
70    SAV=TOT1-TOT
      WRITE (2,65) SAV
65    FORMAT (/62X,'TOTAL SAVING',F9.2)
      STOP
      END
```

Compare the final version with the version in memory by entering the command B0P to display the first 24 lines of the program and 1P to display the next 24 lines.

```
*B0P                    <ENTER>
      PROGRAM AMORT
C     CALCULATES MONTHLY MORTGAGE PAYMENTS AND
C     TOTAL PAYMENTS OVER TERM OF LOAN
      WRITE (3,10)
10    FORMAT (2X,'ENTER PRINCIPAL')
      READ (3,15) P
15    FORMAT (F7.0)
      WRITE (3,20)
20    FORMAT (2X,'ENTER ANNUAL INTEREST RATE')
      READ (3,25) R
25    FORMAT (F7.2)
      WRITE (3,30)
30    FORMAT (2X,'ENTER TERM IN YEARS')
      READ (3,35) N
35    FORMAT (I3)
      MR=R/12
      PAY=(P*MR)/(1-(1/(1+MR)**(12*N)))
      TOT=PAY*12*N
      WRITE (3,40)
40    FORMAT (30X,'SUMMARY OF MORTGAGE PAYMENTS'//)
      WRITE (3,45)
45    FORMAT (2X,'PRINCIPAL',5X,'INTEREST RATE (%)',5X,
*_
```

```
*1P                     <ENTER>
     X  '# OF YEARS',5X,'MONTHLY PAYMENT',5X,'TOTAL PAYMENT'/)
      WRITE (3,50) P,R,N,PAY,TOT
50    FORMAT (2X,F9.2,10X,F7.2,13X,I3,2(11X,F9.2))
      STOP
      END
*_
```

16. Insert the statement M=N before FORMAT statement 35

```
*B2F35^0LT              <ENTER>
35      FORMAT (I3)
*I        M=N           <ENTER>
        M=N*_
```

17. Seven statements must be inserted before the STOP statement. Rather than use the F command to move the character pointer (CP) to the line containing the STOP statement, an absolute line number may be referenced. Use the V command to prefix absolute line numbers to each line displayed and type from the current line to absolute line number 30.

```
M=N*V:30T                    <ENTER>
17:  35   FORMAT (I3)
18:       MR=R/12
19:       PAY=(P*MR)/(1-(1/(1+MR)**(12*N)))
20:       TOT=PAY*12*N
21:       WRITE (2,40)
22:  40   FORMAT (30X,'SUMMARY OF MORTGAGE PAYMENTS'//)
23:       WRITE (2,45)
24:  45   FORMAT (2X,'PRINCIPAL',5X,'INTEREST RATE (%)',5X,
25:     X  '# OF YEARS',5X,'MONTHLY PAYMENT',5X,'TOTAL PAYMENT'/)
26:       WRITE (2,50) P,R,N,PAY,TOT
27:  50   FORMAT (2X,F9.2,10X,F7.2,13X,I3,2(11X,F9.2))
28:       STOP
29:       END
30:
17: *_
```

Move the character pointer (CP) to absolute line number 28, the STOP statement, and type the line.

```
17: *28:T                    <ENTER>
28:       STOP
28: *_
```

Insert the program statements as shown below.

```
28: *I                       <ENTER>
_ 28:
```

NOTE

The next absolute line number is displayed each time a program statement is inserted in the memory buffer. When the TAB key is depressed prior to entering the next line, this absolute line number is erased.

```
          IF (N.NE.M) GO TO 70
          TOT1=TOT
          N=N-2
          GO TO 55
70        SAV=TOT1-TOT
          WRITE (2,65) SAV
65        FORMAT (/62X,'TOTAL SAVING',F9.2)

          35:                        <CTRL-Z>
          35: *_
```

18. Absolute line numbers 21 through 25 must be moved using the X command so that they will be executed before absolute line number 18. Move the CP to absolute line number 21 and type the line.

```
35:  *21:T                <ENTER>
21:            WRITE (2,40)
21:  *_
```

Using the block move command (X), transfer the current line and the next four lines to the temporary LIB file.

```
21:  *5X                  <ENTER>
21:  *_
```

Delete these five lines from the memory buffer and then move the CP back four lines and type the next eight lines.

```
21:  *5K-4L8T             <ENTER>
17:   35       FORMAT (I3)
18:            MR=R/12
19:            PAY=(P*MR)/(1-(1/(1+MR)**(12*N)))
20:            TOT=PAY*12*N
21:            WRITE (2,50)P,R,N,PAY,TOT
22:   50       FORMAT (2X,F9.1,10X,F7.2,13X,I3,2(11X,F9.2))
23:            IF (N.NE.M) GO TO 70
24:            TOT1=TOT
17:  *_
```

Move the CP to absolute line number 18, type the line and retrieve the LIB file containing the five lines deleted in step 21. These lines will be inserted in the memory buffer before absolute line number 18.

```
17:  *18:TR               <ENTER>
18:            MR=R/12
23:  *_
```

Move the CP to absolute line number 17 and type through line number 25.

```
23: *17::25T              <ENTER>
17:   35   FORMAT (I3)
18:        WRITE (2,40)
19:   40   FORMAT (30X,'SUMMARY OF MORTGAGE PAYMENTS'//)
20:        WRITE (2,45)
21:   45   FORMAT (2X,'PRINCIPAL',5X,'INTEREST RATE (%)',5X,
22:      X  '# OF YEARS',5X,'MONTHLY PAYMENT',5X,'TOTAL PAYMENT'/)
23:        MR=R/12
24:        PAY=(P*MR)/(1-(1/(1+MR)**(12*N)))
25:        TOT=PAY*12*N
17: *_
```

19. Insert statement label 55 in absolute line number 24.

```
     17:  *24:I55^              <ENTER>
55     24: *0LT                 <ENTER>
     24:  55          PAY=(P*MR)/(1-(1/(1+MR)**(12*N)))
     24:  *_
```

NOTE

The ^ symbol in the above command line indicates the depression of CTRL-Z.

20. Move the CP to absolute line number 16 and type from the current line to absolute line number 37. This will display all modifications made following the H command entered in step 14.

```
24: *16::37T                    <ENTER>
16:       READ (3,35) N
17:       M=N
18:  35   FORMAT (I3)
19:       WRITE (2,40)
20:  40   FORMAT (30X,'SUMMARY OF MORTGAGE PAYMENTS'//)
21:       WRITE (2,45)
22:  45   FORMAT (2X,'PRINCIPAL',5X,'INTEREST RATE (%)',5X,
23:     X  '# OF YEARS',5X,'MONTHLY PAYMENT',5X,'TOTAL PAYMENT'/)
24:       MR=R/12
25:  55   PAY=(P*MR)/(1-(1/(1+MR)**(12*N)))
26:       TOT=PAY*12*N
27:       WRITE (2,50) P,R,N,PAY,TOT
28:  50   FORMAT (2X,F9.2,10X,F7.2,13X,I3,2(11X,F9.2))
29:       IF (N.NE.M) GO TO 70
30:       TOT1=TOT
31:       N=N-2
32:       GO TO 55
33:  70   SAV=TOT1-TOT
34:       WRITE (2,65) SAV
35:  65   FORMAT (/62X,'TOTAL SAVING',F9.2)
36:       STOP
37:       END
16: *_
```

21. End the edit. Rename the backup file AMYR.FOR

```
   16:  *E                      <ENTER>

A>REN AMYR.FOR = AMORT.BAK
A>
```

Refer to the ADDS*FORTRAN™ Reference Manual for instructions for using the FORTRAN Compiler to create a relocatable FORTRAN module. See Section 7 of this manual for instructions for using the Linking Loader to link and execute this relocatable module. Sample output appears on the following page.

SUMMARY OF MORTGAGE PAYMENTS

PRINCIPAL	INTEREST RATE (%)	# OF YEARS	MONTHLY PAYMENT	TOTAL PAYMENT
20000.00	.08	20	167.29	40149.13
20000.00	.08	18	174.99	37798.41
			TOTAL SAVING	2350.72

SECTION 5:

INTERFACE GUIDE

INTRODUCTION

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

ADOS Organization

ADOS is logically divided into four parts:

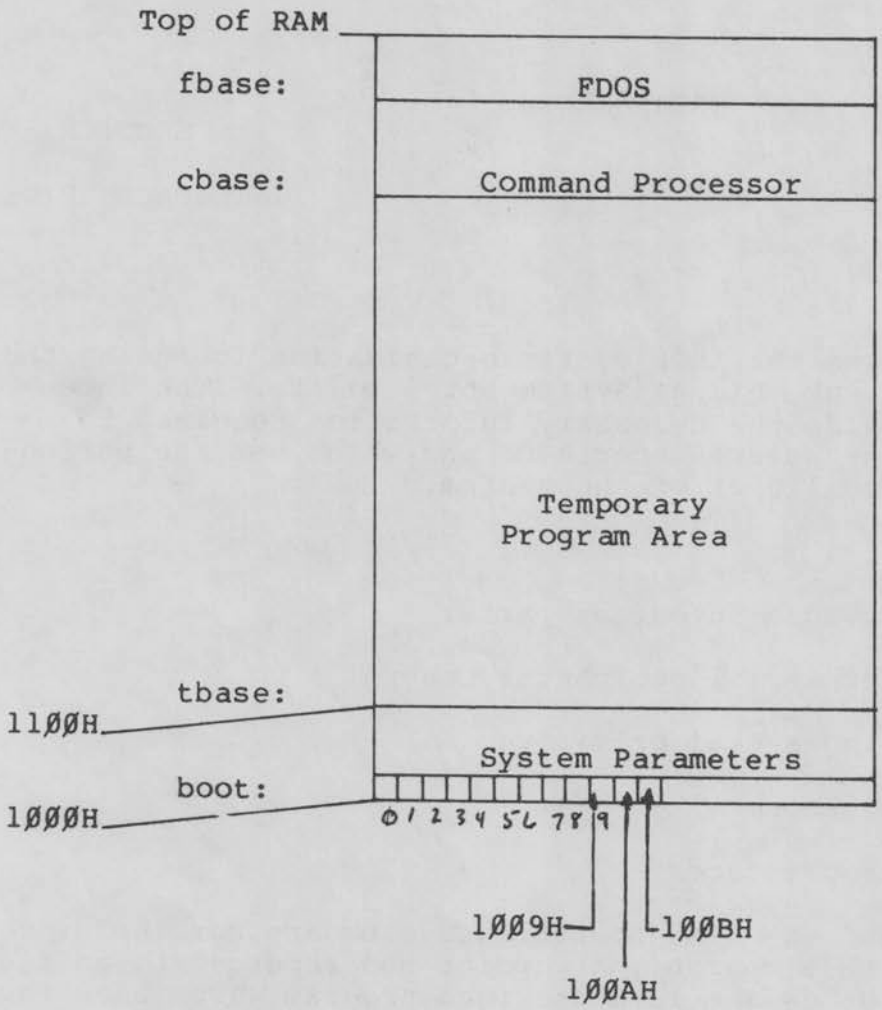
- I/O subsystem for serial peripheral control
- the disk operating system primitives
- the Command Processor
- the temporary program area

The I/O subsystem and the disk operating system are combined into a single program with a common entry point and referred to as the FDOS. The Command Processor is a distinct program which uses the FDOS to provide a human-oriented interface to the information which is catalogued on the diskette. The temporary program area is an area of memory (i.e., the portion which is not used by the FDOS and Command Processor) where various non-resident operating system commands are executed. User programs also execute in the temporary program area. The organization of memory in a standard ADOS system is shown in Figure 5-1.

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

Operation of Transient Programs

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



address field of jump is fbase

entry: the principal entry point to FDOS is at location 1ØØ9H which contains a JMP to fbase. The address field at location 1ØØAH (low order) and 1ØØBH (high order) can be used to determine the size of available memory, assuming the Command Processor is being overlaid.

Figure 5-1. Memory Organization

NOTE

The exact addresses for boot, tbase, cbase, fbase, and entry vary with the ADOS version.

```
<command>
<command>  <filename>
<command>  <filename>.<filetype>
```

where <command> is either a built-in command (e.g., DIR or TYPE), or the name of a transient command or program. If the <command> is a built-in function of ADOS, it is executed immediately; otherwise the Command Processor 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 temporary program area, and thus implicitly originates at tbase in memory. The Command Processor loads the COM file from the diskette into memory starting at tbase, and extending up to address cbase.

If the <command> is followed by either a <filename> or <filename>.<filetype>, then the Command Processor prepares a file control block (FCB) in the system information area of memory (105CH). This FCB is in the form required to access the file through the FDOS, and is given in detail in Table 5.2.

The program then executes, perhaps using the I/O facilities of the FDOS. If the program uses no FDOS facilities, then the entire remaining memory area is available for data used by the program. If the FDOS is to remain in memory, then the transient program can use only up to location fbase as data.* In any case, if the Command Processor area is used by the transient, the entire ADOS system must be reloaded upon the transient's completion. This system reload is accomplished by a direct branch to location "boot" in memory (1ØØØH).

The transient uses the ADOS I/O facilities to communicate with the operator's console and peripheral devices, including the floppy disk subsystem. The I/O system is accessed by passing a "function number" and an "information address" to ADOS through the address marked "entry" in Figure 5-1. In the case of a disk read, for example, the transient program sends the number corresponding to a disk read, along with the address of an FCB, and ADOS performs the operation, returning with either a disk read complete indication or an error number indicating that the disk operation was unsuccessful. The function numbers and error indicators are given in detail in Table 5.3.

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

Operating System Facilities

ADOS facilities which are available to transients are divided into two categories: I/O operations, and DOS (disk operating system) primitives. The I/O operations are listed first:*

Interrogate Console Status
Print Console Buffer
Read Console Buffer
Read Console Character
Write Console Characer
Write List Device Character

The exact details of I/O access are given below. The DOS primitives include the following operations:

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

The details of DOS access are given in Disk I/O Facilities, page 5-7.

BASIC I/O FACILITIES

Access to common peripherals is accomplished by passing a function number and information address to the I/O subsystem. In general, the function number is passed in Register C, while the information address is passed in Register pair D,E.

Direct and Buffered I/O

The I/O subsystem entry points are given in Table 5.1. In the case of simple character I/O to the console, the I/O subsystem reads the console device, and removes the parity bit. The character is echoed back to the console, and tab characters (CTRL-I) are expanded to tab positions starting at column one and separated modulo eight character positions. The buffered read operation takes advantage of the ADOS line editing facilities.

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

That is, the program sends the address of a read buffer whose first byte is the length of the buffer. The second byte is initially empty, but is filled in by ADOS to the number of characters read from the console after the operation (not including the terminating ENTER). The remaining positions are used to hold the characters read from the console. The line editing functions which are performed during this operation are given below:

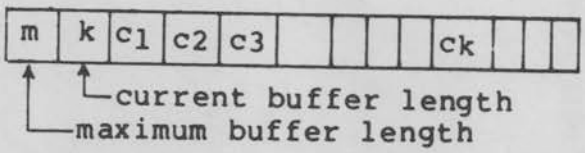
ERASE	- deletes characters from the current cursor position to the end of the line.
DELETE CHAR	- deletes the character at the current cursor position and left justifies to fill the space left by the deleted character.
INSERT CHAR	- permits one or more characters to be inserted at the current cursor position.
CURSOR CONTROL	- left-arrow and right-arrow keys allow free backward and forward movement of the cursor within the line. The HOME key returns the cursor to the first character of the line.
RESET	- system warm boot reads the directory but does not reload ADOS.
NEW LINE	- return carriage, but do not accept buffer (physical carriage return)
ENTER	- accept buffer

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

Table 5.1. Basic I/O Operations

Reg. C Function Number	Reg. D-E Entry Parameters	Reg. A-B Returned Value	Typical Call
Read Console 1	None	ASCII Character	MVI C,1 CALL ENTRY
Write Console 2	ASCII Character	None	MVI C,2 MVI E,CHAR CALL ENTRY
Write List 5	ASCII Character	None	MVI C,5 MVI E,CHAR CALL ENTRY
Print Buffer 9	Address of string terminated by '$'	None	LXI D,MESSAGE MVI C,9 CALL ENTRY . . . MESSAGE: DB 'HELLO $'
Read Buffer 10	Address of Read Buffer*	Read buffer is filled to maximum length with console characters	LXI D,BUFFER MVI C,IØ CALL ENTRY
Interrogate Console Ready 11	None	Byte value with least significant bit = 1 (true) if console character is ready	MVI C,11 CALL ENTRY

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



DISK I/O FACILITIES

The DOS section of ADOS provides access to files stored on diskettes. The discussion which follows gives the overall file organization, along with file access mechanisms.

File Organization

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

BAK	Backup File produced by Editor (see Section 4, Editor)
BAS	BASIC Source File (ASCII or Binary compressed)
COM	Memory Image (object) File (i.e., "Command" file for transients.)
CRF	Cross Reference File
FOR	FORTRAN Source File
MAC	Assembler Source File
PRN	Assembler Listing File
REL	FORTRAN or Assembler Relocatable File
SUB	Exec Processor File for SUBMIT
$$$	Temporary File created and normally erased by Editor and Utilities

Thus, the name

X.MAC

is interpreted as an assembly language source file by the Command Processor with <filename> X. A complete list of generic filetypes appears in Table 3.1.

The files in ADOS are organized as a logically contiguous sequence of 128 byte records, which are normally read or written in sequential order. Random access is allowed under ADOS, however. No particular format within records is assumed by ADOS, although some transients expect particular formats:

1. Source files are considered a sequence of ASCII characters, where each "line" of the source file is followed by carriage-return-line-feed characters. Thus, one 128 byte

ADOS record could contain several logical lines of source text. End of text is given by the character CTRL-Z, or real end-of-file returned by ADOS.

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

File Control Block Format

Each file accessed must have a file control block associated with it. The file control block or FCB contains the name and all location information for all file accesses.

The FCB is a 33-byte area of RAM. The FCB format is given in Table 5.2. When initially accessing ADOS files, it is the programmer's responsibility to fill the lower 16 bytes of the FCB, along with the NR field. Normally, the FN and FT fields are set to the ASCII filename and filetype while all other fields are set to zero. Each FCB describes up to 16K bytes of a particular file (0 to 128 (sectors) records of 128 bytes each), and, using automatic mechanisms of ADOS, up to 15 additional extensions of the file can be addressed. Thus, each FCB can potentially describe files up to 256K bytes (which is slightly larger than the diskette capacity).

FCBs are stored in a directory area of the diskette (partition Ø1), and are brought into central memory before file operations by the OPEN or SEARCH command then updated in memory as file operations proceed, and finally recorded on the diskette at the termination of the file operation by the CLOSE command. This organization makes the file highly reliable, since diskette file integrity can only be disrupted in the unlikely case of hardware failure during update of a single directory entry.

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

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

PROGNAME X.AOT Y.BOP

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

If no file names were specified in the original command, then the fields beginning at tfcb and tfcb + 16 both contain blanks (2ØH). If one file name was specified, then the field at tfcb + 16 contains blanks. If the filetype is omitted, then the field is assumed to contain blanks. In all cases, the **Command Processor** translates lower case alphabetics to upper case to be consistent with the ADOS file naming conventions.

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

```
tbuff:

+0 +1 +2 +3 +4 +5 +6 +7 +8 +9 +10 +11 +12 +13 +14 +15

12 Ø  X  .  A  O  T  Ø  Y  .  B  O  P  ?  ?  ?
```

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

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

In a standard ADOS system, the following values are assumed:

boot:	1ØØØH	bootstrap load (warm start)
entry:	1ØØ9H	entry point to FDOS
tfcb:	1Ø5CH	first default file control block
tfcb+16	1Ø6CH	second file name
tbuff	1Ø8ØH	default buffer address
tbase:	11ØØH	base of transient area

File Access

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

$$e = \frac{r}{128}$$

or equivalently, $e = SHR\ (r,7)$

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

$$N = r \bmod 128$$

or

$$N = r\ AND\ 7FH.$$

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

Partition Disk Address Calculation

ADOS allocates diskette space to files in whole partition increments only. The current single-sided floppy diskette version defines a partition as one kilobyte of storage space. Partitions are comprised of 8 sequential sectors on the diskette. Where a partition crosses a track boundary, sector numbering is still contiguous, i.e., 24, 25, 26, 1, 2, 3, 4. There are 242 partitioned areas on the diskette (all the space on the diskette from track 2-76). The first two partitions (number Ø and 1) are reserved for the directory and therefore can never be allocated to a file by ADOS. The remaining 240 partitions are assigned to an open file on a demand basis. The formula to determine the track number of a given partition would be:

$$TRACK = \frac{PARTITION}{Y} * X + 2$$

The sector would be calculated as:

```
SECTOR = (PARTITION*X) MOD Y + 1
```

where

```
X = PARTITION SIZE/SECTOR SIZE
Y = SECTORS PER TRACK
```

In the current single-sided floppy disk version, X=8 and Y=26.

Table 5.2. File Control Block (FCB) Field Definitions

Position	Field	Description
Ø	ET	Entry Type - ØØ = Active E5 = Not Active
1-8	FN	File Name - 8-byte ASCII name, left-justified. If size is less than 8 bytes, field is padded with ASCII space codes.
9-11	FT	File Type - 3-byte ASCII extension, left-justified. Unused positions are padded with ASCII space codes.
12	EX	File Extent - Binary value of Ø-15. Each extent represents a file segment of up to 16K bytes or 16 partitions (a partition is equivalent to a 1K block allocation). A file larger than 16K will require another FCB with the EX field incremented. The first extent is always Ø. Since each extent can contain 16K bytes and up to 16 extents may be specified, a file of 16 x 16 = 256K bytes is possible. On present media this size is slightly larger than a full diskette.
13-14		Unused, assume zero.
15	RC	Record Count - Current extent size (0-128). An extent may contain 16 partitions. Each partition is 1K bytes and represents 8 contiguous 128 byte sectors. Therefore, maximum record count is equivalent to 16 x 8 = 128 records or sectors.
16-31	DM	Disc Allocation Map - Each of these 16 bytes indicates the partition number allocated for this extent. The partition number is obtained by ADOS from the "Free Space Map."
32	NR	Next Record - Indicates to disc access primitives which record (sector) to read or write within this extent. This number must be within the range 1 to 128 since a maximum of 128 records (sectors) may be contained in one extent.

Table 5.3. ADOS Disk Access Primitives

Function	Function # In C Reg	Entry Parameters	Returning Parameters	Comments
LOGIN/ SELECT	14	D = DRV NO. (BINARY)	None	If LOGIN, first call after a boot, Free Space Map is created. Specified drive will be "ON-LINE".
OPEN	15	DE = address of FCB. FN, FT & EX fields of FCB must be set up.	A = relative directory position of file, if found. If A = 255, no file is present.	OPEN is only used if a file exists. The extent to OPEN is specified by the entry parameters. The DM field of the FCB is set up.
CLOSE	16	DE = address of previously opened or created FCB.	" "	
SEARCH	17	DE = address of FCB with FN & FT fields set up. Wild card "?" feature is supported.	A = relative directory position of first file, if found. If A = 255, no file is present.	If a file is found, it is not opened. Only extents of zero are searched. Note: Directory entry is returned in default buffer (tbuff). Four directory entries are read each time. User must offset by the A register into tbuff to determine the address of entry.
SEARCH FOR NEXT OCCURRENCE	18	DE = address of FCB used in search (above) primitive.	A = relative directory position of next file occurrence. If A = 255, no other matches found.	This command must be used after first calling SEARCH. Note: No intervening system I/O calls allowed between SEARCH and SEARCH NEXT.

Table 5.3. ADOS Disk Access Primitives (continued)

Function	Function # In C Reg	Entry Parameters	Returning Parameters	Comments
DELETE	19	DE = address of FCB, FN & FT set up.	None	All extents are deleted. Only the ET fields of the directory entries modified changed to non-active. Should use SEARCH first to insure presence of file in directory.
READ NEXT	20	DE = address of FCB which has been successfully OPENED. NR field is set to record (sector) to read.	A = Ø successful read; A = 1, read past end of file. A = 2, reading unwritten data (NR) = (NR + 1), if (NR) = 128 next extent is opened at (NR) = 1.	Read will transfer the data to the previously set DMA address. (function 26)
WRITE NEXT	21	DE = address of FCB which has been successfully OPENED. NR field is set to record (sector) to write.	A = Ø successful write; A = 1, error in extending file. A = 2, end of disk error. A = 255, no more directory space.	Write will transfer the data from the previously set DMA address. (function 26)
CREATE	22	DE = address of FCB FN & FT set up.	A = relative directory position of file or 255 if no directory space is available.	CREATE is only used if no file exists. If in doubt, do SEARCH operation first.
RENAME	23	DE = address of FCB with old FN & FT in first 16 bytes and new FN & FT in second 16 bytes.	A = relative directory position of first match of old FN & FT. If A = 255, no match was found.	All extents of old FN & FT are modified.

Table 5.3. ADOS Disk Access Primitives (continued)

Function	Function # In C Reg	Entry Parameters	Returning Parameters	Comments
INTERROGATE LOGIN VECTOR	24	None	A = byte value with "1" in bit positions of "on-line" disk, disk A is at bit position Ø, B at bit position 1, etc.	
INTERROGATE DRIVE NO.	25	None	A = disk number of currently logged disk.	All disk access is from logged disk.
Set DMA ADDRESS	26	DE = DMA ADDRESS	INTERNAL DMA POINTER SET	All future disk accesses will be to and from the set address (buffer size is 128 bytes). ADOS internally sets this to the value 1Ø8ØH every warm or cold boot.

GENERAL NOTES

1) All register values are changed during a primitive call.

2) Disk hardware failures are reported to the console by ADOS. The primitive command being executed during a disk failure will be bypassed (not performed) upon a hardware failure.

ADOS ENTRY POINT SUMMARY

The functions shown below summarize the functions of the FDOS. The function number is passed in Register C and the information is passed in Register D,E. Single byte results are returned in Register A. If a double byte result is returned, then the high-order byte comes back in Register B. The transient program enters the FDOS through location "entry" (see Sample Programs) in assembly language. All registers are altered in the FDOS.

Table 5.4. FDOS Functions

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

ADDRESS ASSIGNMENTS

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

The address assignments are

```
boot  = 1000H   warm start operation
tfcb  = 105CH   default file control block location
tbuff = 1080H   default buffer location
tbase = 1100H   base of transient program area
entry = 1009H   entry point to disk system from user programs
```

SAMPLE PROGRAM

This section contains a sample program that will interface with the ADOS operating system. The program is written in assembly language, and is the source program for the DUMP utility.

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

```
ENTRY   EQU   1009H
```

for example, gives the ADOS entry point for peripheral I/O functions. The default file control block address is also defined (FCB), along with the default buffer address (BUFF). Note that the program is set up to run at location 1100H which is the base of the transient program area. The stack is first set up by saving the entry stack pointer into OLDSP, and resetting SP to the local stack. The stack pointer upon entry belongs to the Command Processor, and need not be saved unless control is to return to ADOS upon exit. That is, if the program terminates with a reboot (branch to location 1000H) then the entry stack pointer need not be saved.

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

BREAK - when called, checks to see if there is a keyboard character ready. BREAK is used to stop the listing at the CRT

PCHAR - print the character which is in register A at the CRT

CRLF - send carriage return and line feed to the CRT

PNIB - print the hexadecimal value in register A in ASCII at the CRT

PHEX - print the byte value (two ASCII characters) in register A at the CRT

ERR - print error flag #n at the CRT, where n is

1 if file cannot be opened
2 if disk read error occurred

GNB - get next byte of data from the input file. If the IBP (input buffer pointer) exceeds the size of the input buffer, then another disk record of 128 bytes is read. Otherwise, the next character in the buffer is returned. IBP is updated to point to the next characer.

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

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

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

until the end of file.

The label FINIS gets control upon end of file. CRLF is called first to return the carriage from the last line output. The Command Processor stack pointer is then reclaimed from OLDSP, followed by a RET to return to the Command Processor. Note that a JMP 1000H could be used following the FINIS label, which would cause the ADOS system to be brought in again from the diskette (this operation is necessary only if the Command Processor has been overlaid by data areas).

The file control block format is then listed (FCBDN ... FCBLN) which overlays the fcb at location 105CH which is set up by the

Command Processor when the DUMP program is initiated. That is, if the user types

```
DUMP  X.Y
```

then the Command Processor sets up a properly formed fcb at location 105CH for the DUMP (or any other) program when it goes into execution. Thus, the SETUP subroutine simply addresses this default fcb, and calls the disk system to open it. The DISKR (disk read) routine is called whenever GNB needs another buffer full of data. The default buffer at location 1080H is used, along with a pointer (IBP) which counts bytes as they are processed. Normally, an end of file condition is taken as either an ASCII 1AH (CTRL-Z), or an end of file detection by the DOS. The file dump program, however, stops only on a DOS end of file.

```
                ;       FILE DUMP PROGRAM, READS AN INPUT FILE AND PRINTS IN HEX
                ;
                ;
1100                    ORG     1100H
1009 =          BDOS    EQU     1009H   ;DOS ENTRY POINT
000F =          OPENF   EQU     15      ;FILE OPEN
0014 =          READF   EQU     20      ;READ FUNCTION
0002 =          TYPEF   EQU     2       ;TYPE FUNCTION
0001 =          CONS    EQU     1       ;READ CONSOLE
000B =          BRKF    EQU     11      ;BREAK KEY FUNCTION (TRUE IF CHAR READY)
                ;
105C =          FCB     EQU     105CH   ;FILE CONTROL BLOCK ADDRESS
1080 =          BUFF    EQU     1080H   ;INPUT DISK BUFFER ADDRESS
                ;
                ;       SET UP STACK
1100 210000             LXI     H,0
1103 39                 DAD     SP
1104 220F11             SHLD    OLDSP
1107 315111             LXI     SP,STKTOP
110A C3C411             JMP     MAIN
                ;       VARIABLES
110D            IBP:    DS      2       ;INPUT BUFFER POINTER
                ;
                ;       STACK AREA
110F            OLDSP:  DS      2
1111            STACK:  DS      64
1151 =          STKTOP  EQU     $
                ;
                ;       SUBROUTINES
                ;
                ;       CHECK BREAK KEY (ACTUALLY ANY KEY WILL DO)
1151 E5         BREAK:  PUSH H
1152 D5                 PUSH D
1153 C5                 PUSH B          ;ENVIRONMENT SAVED
1154 0E0B               MVI     C,BRKF
1156 CD0910             CALL    BDOS
1159 C1                 POP B
115A D1                 POP D
115B E1                 POP H           ;ENVIRONMENT RESTORED
115C C9                 RET
                ;
                ;       PRINT A CHARACTER
115D E5         PCHAR:  PUSH H
115E D5                 PUSH D
115F C5                 PUSH B          ;SAVED
1160 0E02               MVI     C,TYPEF
1162 5F                 MOV     E,A
1163 CD0910             CALL    BDOS
1166 C1                 POP B
1167 D1                 POP D
1168 E1                 POP H           ;RESTORED
1169 C9                 RET
                ;
116A 3E0D       CRLF:   MVI     A,0DH
116C CD5D11             CALL    PCHR
```

```
116F 3E0A           MVI     A,0AH
1171 CD5D11         CALL    PCHAR
1174 C9             RET
                    ;
                    ;
                    ;       PRINT NIBBLE IN REG A
1175 E60F           PNIB:   ANI     0FH     ;LOW 4 BITS
1177 FE0A           CPI     10
1179 D28111         JNC     P10
                    ;       LESS THAN OR EQUAL TO 9
117C C630           ADI     '0'
117E C38311         JMP     PRN
                    ;
                    ;       GREATER OR EQUAL TO 10
1181 C637           P10:    ADI     'A' - 10
1183 CD5D11         PRN:    CALL    PCHAR
1186 C9             RET
                    ;
                    ;       PRINT HEX CHAR IN REG A
1187 F5             PHEX:   PUSH    PSW
1188 0F             RRC
1189 0F             RRC
118A 0F             RRC
118B 0F             RRC
118C CD7511         CALL    PNIB    ;PRINT NIBBLE
118F F1             POP     PSW
1190 CD7511         CALL    PNIB
1193 C9             RET
                    ;
                    ;       PRINT ERROR MESSAGE
1194 CD6A11         ERR:    CALL    CRLF
1197 3E23           MVI     A,'#'
1199 CD5D11         CALL    PCHAR
119C 78             MOV     A,B
119D C630           ADI     '0'
119F CD5D11         CALL    PCHAR
11A2 CD6A11         CALL    CRLF
11A5 C3F711         JMP     FINIS
                    ;
                    ;       GET NEXT BYTE
11A8 3A0D11         GNB:    LDA     IBP
11AB FE80           CPI     80H
11AD C2B411         JNZ     GO
                    ;       READ ANOTHER BUFFER
                    ;
11B0 CD1612         CALL    DISKR
11B3 AF             XRA     A
                    ;
                    ;       READ THE BYTE AT BUFF+REG A
11B4 5F             GO:     MOV     E,A
11B5 1600           MVI     D,0
11B7 3C             INR     A
11B8 320D11         STA     IBP
                    ;       POINTER IS INCREMENTED
                    ;       SAVE THE CURRENT FILE ADDRESS
```

```
11BB E5              PUSH    H
11BC 218010          LXI     H,BUFF
11BF 19              DAD     D
11C0 7E              MOV     A,M
                ;    BYTE IS IN THE ACCUMULATOR
                ;
                ;    RESTORE FILE ADDRESS AND INCREMENT
11C1 E1              POP     H
11C2 23              INX     H
11C3 C9              RET
                ;
                ;            READ AND PRINT SUCCESSIVE BUFFERS
11C4 CDFF11  MAIN:   CALL    SETUP   ;SET UP INPUT FILE
11C7 3E80            MVI     A,80H
11C9 320D11          STA     IBP     ;SET BUFFER POINTER TO 80H
11CC 21FFFF          LXI     H,0FFFFH        ;SET TO -1 TO START
                ;
11CF CDA811  GLOOP:  CALL    GNB
11D2 47              MOV     B,A
                ;    PRINT HEX VALUES
                     SAVE
                ;    CHECK FOR LINE FOLD
11D3 7D              MOV     A,L
11D4 E60F            ANI     0FH     ;CHECK LOW 4 BITS
11D6 C2EB11          JNZ     NONUM
                ;    PRINT LINE NUMBER
11D9 CD6A11          CALL    CRLF
                ;
                ;    CHECK FOR BREAK KEY
11DC CD5111          CALL    BREAK
11DF 0F              RRC
11E0 DAF711          JC      FINISH  ;DON'T PRINT ANY MORE
                ;
11E3 7C              MOV     A,H
11E4 CD8711          CALL    PHEX
11E7 7D              MOV     A,L
11E8 CD8711          CALL    PHEX
11EB 3E20    NONUM:  MVI     A,' '
11ED CD5D11          CALL    PCHAR
11F0 78              MOV     A,B
11F1 CD8711          CALL    PHEX
                     RESTORE
11F4 C3CF11          JMP     GLOOP
                ;
                ;    END PSA
                ;    END OF INPUT
11F7 CD6A11  FINIS:  CALL    CRLF
11FA 2A0F11          LHLD    OLDSP
11FD F9              SPHL
11FE C9              RET
                ;
                ;
                ;    FILE CONTROL BLOCK DEFINITIONS
```

```
105C =          FCBDN   EQU     FCB+0   ;DISK NAME
105D =          FCBFN   EQU     FCB+1   ;FILE NAME
1065 =          FCBFT   EQU     FCB+9   ;DISK FILE TYPE (3 CHARACTERS)
1068 =          FCBRL   EQU     FCB+12  ;FILE'S CURRENT REEL NUMBER
106B =          FCBRC   EQU     FCB+15  ;FILE'S RECORD COUNT (0 to 128)
107C =          FCBCR   EQU     FCB+32  ;CURRENT (NEXT) RECORD NUMBER (0 to 127)
107D =          FCBLN   EQU     FCB+33  ;FCB LENGTH
                ;
                ;
                ;       SET UP FILE
                ;       OPEN THE FILE FOR INPUT
11FF 115C10     SETUP:  LXI     D,FCB
1202 0E0F               MVI     C,OPENF
1204 CD0910             CALL    BDOS
                ;       CHECK FOR ERRORS
1207 FEFF               CPI     255
1209 C21112             JNZ     OPNOK
                ;       BAD OPEN
120C 0601               MVI     B,1     ;OPEN ERROR
120E CD9411             CALL    ERR
                ;
                ;       OPEN IS OK.
1211 AF         OPNOK:  XRA     A
1212 327C10             STA     FCBCR
1215 C9                 RET
                ;
                ;       READ DISK FILE RECORD
1216 E5         DISKR:  PUSH H
1217 D5                 PUSH D
1218 C5                 PUSH B
1219 115C10             LXI     D,FCB
121C 0E14               MVI     C,READF
121E CD0910             CALL    BDOS
1221 C1                 POP B
1222 D1                 POP D
1223 E1                 POP H
1224 FE00               CPI     0       ;CHECK FOR ERRS
1226 C8                 RZ
                ;       MAY BE EOF
1227 FE01               CPI     1
1229 CAF711             JZ      FINIS
                ;
122C 0602               MVI     B,2     ;DISK READ ERROR
122E CD9411             CALL    ERR
                ;
1231                    END
```

1009 BDOS	1151 BREAK	000B BRKF	1080 BUFF	0001 CONS
116A CRLF	1216 DISKR	11F7 EPSA	1194 ERR	105C FCB
107C FCBCR	105C FCBDN	105D FCBFN	1065 FCBFT	107D FCBLN
106B FCBRC	1068 FCBRL	11F7 FINIS	11B4 GO	11CF GLOOP
11A8 GNB	110D IBP	11C4 MAIN	11EB NONUM	110F OLDSP
000F OPENF	1211 OPNOK	1181 P10	115D PCHAR	1187 PHEX
1175 PNIB	1183 PRN	0014 READF	11F4 RESTORE	11D3 SAVE
11FF SETUP	1111 STACK	1151 STKTOP	0002 TYPEF	

SECTION 6:

ASSEMBLER

The computer manipulates data and transfers information between its registers, memory locations and input/output devices under the direction of instructions stored in its memory. These instructions are coded in binary and normally specify the data or the address of the data to be handled, as well as the operation to be performed.

Although it is possible to write programs in binary machine code, the problems of remembering long strings of binary numbers and entering them into the computer without error are practically insuperable in all but the very simplest programs. For this reason, higher-level language translation programs are provided for most computers.

An Assembler is a computer program which takes as input symbolic instructions and addresses and produces machine code programs as output. Instead of having to remember machine language instructions in binary, the programmer can refer to instructions by symbolic names which are easier to remember. For example, the first location in a routine which calculates could be CALC and subsequent references to the name CALC would refer to that address.

Assemblers also provide system aids such as listings of source programs, cross reference listings and relocatable code (which may be loaded and executed anywhere in memory).

The ADOS Assembler, compatible with INTEL's 8080 Assembler, is a two-pass Assembler. On the first pass, it reads the source program and assigns addresses to labels. It then reads the program again and translates the symbolic instructions into binary code. At this time, a listing of the source code may be printed, or a print file may be created (if requested). The output of the second pass is a relocatable object code "LOAD" module which may be loaded anywhere in memory by the LINKER and executed.

The user should refer to the INTEL Assembler manual for all details regarding Assembly programs.

GENERAL COMMAND STRUCTURE

The Assembler is loaded and run using one of the following forms of the command when the **Command Processor** is in control:

```
A<M80 <ENTER>
A<M80 Command Line <ENTER>
```

In the first form, the Command Lines are read directly from the keyboard prompted with the asterisk (*). After the last Command Line is entered control is returned to the **Command Processor** when RESET or CTRL-C is depressed. The second form is equivalent to the first, except that the Command Line is executed immediately and control is automatically returned to the **Command Processor**.

The form of the Command Line determines which files and listings will be produced.

In the general forms of the Command Line below, the following abbreviations are used:

drv:	is the disk drive to be used; if omitted, the default is the currently logged-in drive.
filnam.MAC	is the assembly language (.MAC) source file (filnam) to be assembled.
filnam.REL	is the relocatable (.REL) module produced by the Assembler from the specified file (filnam).
filnam.PRN	is the print file (.PRN) produced by the Assembler from the specified file (filnam). The side-by-side Assembler listing contains a list of error codes, machine language code, assembly language code and a symbol table. (See pages 6-5 and 6-6 for an example of Assembler output.)
LST:	indicates the attached printer.
CON:	indicates the screen.
swl-swn	are the option switches which are described below.

The general forms of the Command Line are shown below:

```
*[drv:]filnam[.REL],[drv:]filnam[.PRN]=
                          [drv:]filnam[.MAC]/swl/.../swn

                    [LST]:
*[drv:]filnam[.REL],[CON]:=[drv:]filnam[.MAC]/swl/.../swn
```

If the filetypes are omitted, the system defaults to .MAC for the assembly language source file, .REL for the relocatable module and .PRN for the print file. Any three characters may be specified for each filetype. However, if any or all of them differ from the default filetypes for M80 they must be included in the Command Line.

OPTION SWITCHES

Switch	Action
/C	cross reference switch (see page 6-8)
/H	print all listing addresses in hexadecimal
/L	force generation of a listing file
/R	force generation of an object file

EXAMPLES OF ASSEMBLER COMMAND LINES

Example 1

a)*TESTF1.REL,TESTF1.PRN=TESTF1.MAC

b)*TESTF1,TESTF1=TESTF1

Versions a and b are two versions of the same Command Line. They both use the source file, TESTF1.MAC, on the logged-in drive to produce a relocatable file, TESTF1.REL, and a print file, TESTF1.PRN, also on the logged-in drive. A list of errors is generated on the screen. An abbreviated version of a is given in b with the file types assumed to be REL, PRN and MAC, respectively.

The relocatable file and the print file do not have to be saved on the same diskette as the source file. For example:

*B:TESTF1.REL,TESTF1.PRN=TESTF1.MAC

The source file on the logged-in drive is used to produce a print file also on the logged-in drive and a relocatable file on drive B.

Example 2

```
                [LST]
a)*PGM1.REL,[CON]:=PGM1.MAC

          [LST]
b)*PGM1,[CON]:=PGM1
```

In example 2 above, a and b are two versions of the same Command Line. Both versions use the source file, PGM1.MAC, on the logged-in drive to produce a relocatable file, PGM1.REL, on the logged-in drive and a print file which is output on either the attached line printer or the screen, depending on the option chosen. Notice that in version b the file types are assumed to be REL and MAC respectively.

Again, the user has the option of saving the relocatable file and the source file on different disks. For example:

```
*PGM1.REL,LST:=B:PGM1.MAC    or    *PGM1,LST:=B:PGM1
```

The source file on drive B is used to produce a relocatable file on the logged-in drive and a print file which is output to the attached line printer.

Example 3

```
a) *=MYFILE.MAC
b) *=MYFILE
```

In example 3 above, a and b are two versions of the same Command Line. They both use the source file, MYFILE.MAC, on the logged-in drive to produce a relocatable file, MYFILE.REL, on the logged-in drive. A list of errors is generated on the screen. Version b is an abbreviated version of a with the file type assumed to be MAC.

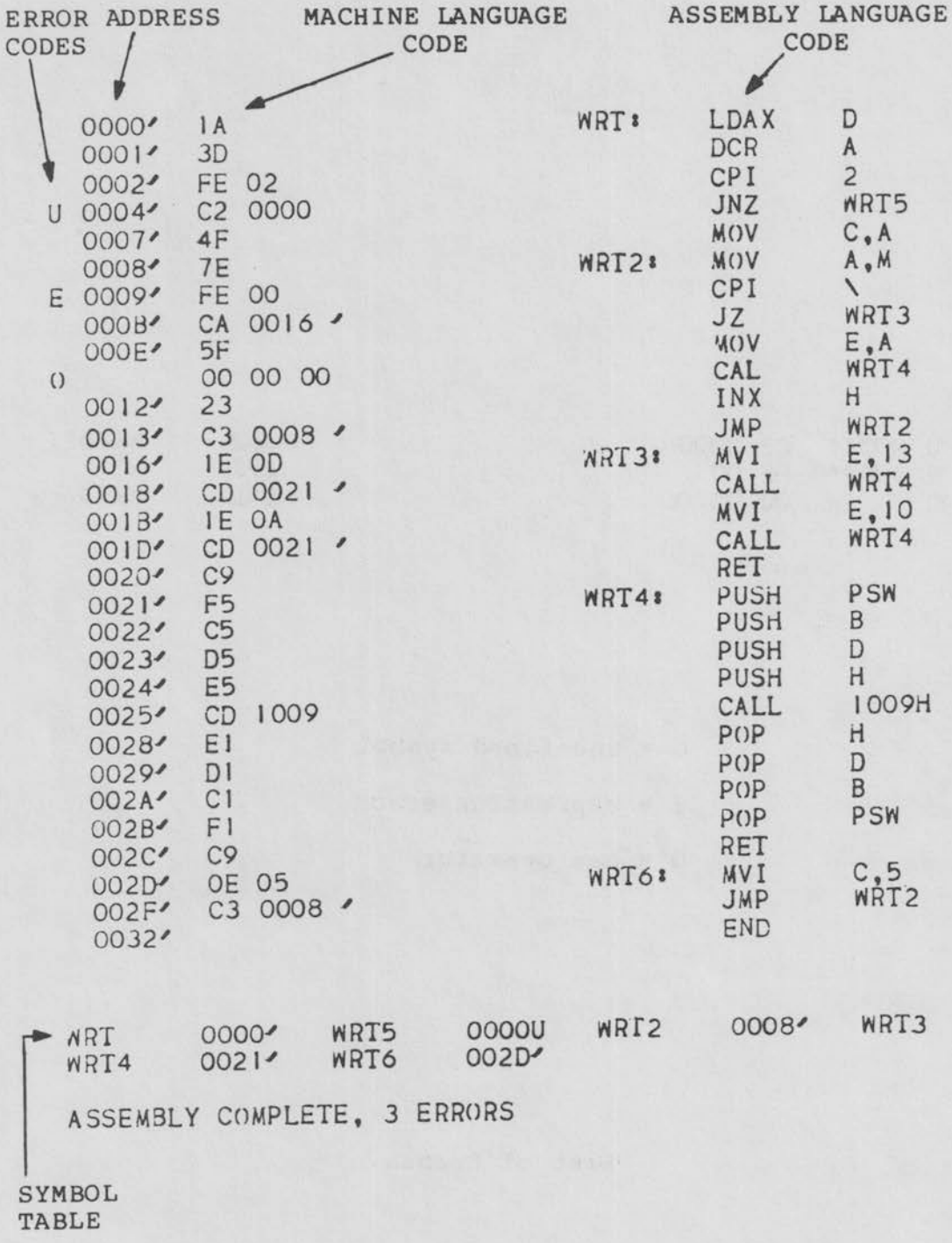
The relocatable file is always saved on the same disk drive as the source file. In the example below, the source file on drive B is used to produce the relocatable file on drive B.

```
*= B:MYFILE.MAC    or    *= B:MYFILE
```

Example 4

```
a) *,= ACCT.MAC
b) *,= ACCT
```

In example 4 above, a and b are again versions of the same Command Line. They both use the source file ACCT.MAC, on the logged-in drive to produce only an error listing on the screen. Version b is an abbreviated version of version a with the file type assumed to be MAC.



Assembler Output

```
U 0004'  C2 0000          JNZ     WRT5
E 0009'  FE 00            CPI     \
O        00 00 00         CAL     WRT4
```

U = undefined symbol

E = expression error

O = bad operator

List of Errors

ASSEMBLER ERRORS

Each line in error is always displayed on the terminal regardless of whether or not a listing is being printed or displayed. Assembler errors are indicated by a one character flag in column one of the listing file. The list of Assembler Error Codes is given below.

Code	Meaning
B	Block Name in DATA
C	Too Many COMMONS
D	Bad Octal or Hex Digit
E	Expression Error
I	Illegal Character
L	No Label in EQU
M	Label or Symbol Defined More Than Once
N	Name Too Long
O	Bad Operator (Opcode)
P	Phase Error
Q	Questionable Syntax
T	Illegal Field Termination
U	Undefined Symbol
V	Value Error to MOD
X	Illegal Operand
Z	Missing Second Field for Opcode

CROSS REFERENCE FACILITY

In order to generate a cross reference listing, the Assembler must output a special listing file with embedded control characters. The cross reference switch, /C, causes the Assembler to open a .CRF file instead of a .PRN file. If the /L switch is also included in the Assembler Command Line, it is ignored. Any other switch will cause an error message to be sent to the screen.

The Cross Reference Facility is invoked by typing one of the following forms of the command

```
A>CREF80  <ENTER>  or
A>CREF80  Command Line  <ENTER>
```

after the Assembler has returned control to the **Command Processor**.

In the first form, the Command Lines are read directly from the keyboard prompted with the asterisk(*). After the last Command Line is entered, control is returned to the **Command Processor** when RESET or CTRL-C is depressed. The second form is equivalent to the first, except that the Command Line is executed immediately and control is automatically returned to the **Command Processor**.

The general form of the Command Line is shown below:

```
*[drv:]filename[.PRN]=filename[.CRF]
```

Cross reference listing files differ from ordinary listing files in that:

1. Each source statement is given a line number.

2. At the end of the listing, variable names appear in alphabetic order along with the numbers of the lines on which they are referenced or defined. The pound sign (#) indicates the line number in the program on which the label is defined.

Examples: Assembler Command Lines with Cross Reference Option Switch /C

A>M80	Invoke the ADOS Assembler.
*WRT,WRT=WRT/C	Assemble file WRT.MAC and create a relocatable file WRT.REL and a cross reference file WRT.CRF
*ACCT,LIST=TEST/C	Assemble file TEST.MAC and create a relocatable file ACCT.REL and a cross reference file LIST.CRF

Examples: Cross Reference Facility Command Lines

A>CREF80	Invoke the Cross Reference Facility.
*=WRT	Examine file WRT.CRF and generate a cross reference listing file WRT.PRN
*CRLST=LIST	Examine file LIST.CRF and generate a cross reference listing file CRLST.PRN

The Assembler source code on page 6-5 has been corrected and re-assembled with the cross reference option switch /C. The cross reference listing file appears on the following pages.

```
LINE     ADDRESS    MACHINE LANGUAGE      ASSEMBLY LANGUAGE
NUMBERS                  CODE                   CODE
   1      0000'                          ENTRY WRT
   2      0000'  1A                WRT:  LDAX    D
   3      0001'  3D                      DCR     A
   4      0002'  FE 02                   CPI     2
   5      0004'  C2 002D '               JNZ     WRT5
   6      0007'  4F                      MOV     C,A
   7      0008'  7E                WRT2: MOV     A,M
   8      0009'  FE 02                   CPI     2
   9      000B'  CA 0016 '               JZ      WRT3
  10      000E'  5F                      MOV     E,A
  11      000F'  CD 0021 '               CALL    WRT4
  12      0012'  23                      INX     H
  13      0013'  C3 0008 '               JMP     WRT2
  14      0016'  1E 0D             WRT3: MVI     E,13
  15      0018'  CD 0021 '               CALL    WRT4
  16      001B'  1E 0A                   MVI     E,10
  17      001D'  CD 0021 '               CALL    WRT4
  18      0020'  C9                      RET
  19      0021'  F5                WRT4: PUSH    PSW
  20      0022'  C5                      PUSH    B
  21      0023'  D5                      PUSH    D
  22      0024'  E5                      PUSH    H
  23      0025'  CD 1009                 CALL    1009H
  24      0028'  E1                      POP     H
  25      0029'  D1                      POP     D
  26      002A'  C1                      POP     B
  27      002B'  F1                      POP     PSW
  28      002C'  C9                      RET
  29      002D'  0E 05             WRT5: MVI     C,5
  30      002F'  C3 0008 '               JMP     WRT2
  31      0032'                          END

WRT     0000'   WRT5    002D'   WRT2    0008'   WRT3    0016'
WRT4    0021'

ASSEMBLY COMPLETE, NO ERRORS
```

SYMBOL TABLE

Cross Reference Listing File

A	3	6	7	10
B	20	26		
C	6	29		
D	2	21	25	
E	10	14	16	
H	12	22	24	
M	7			
PSW	19	27		
WRT	1	2#		
WRT2	7#	13	30	
WRT3	9	14#		
WRT4	11	15	17	19#
WRT5	5	29#		

Label WRT5 is defined on line number 29.

Cross Reference List

SECTION 7:

LINKING LOADER

The Linking Loader loads program modules into memory, calculates addresses for names, resolves external references and executes the resulting code package.

GENERAL COMMAND STRUCTURE

The Linking Loader is loaded and run using one of the following forms of the command when the **Command Processor** is in control:

```
A>L80 <ENTER>      or
A>L80 Command Line <ENTER>
```

With both forms, the Command Lines are read directly from the keyboard prompted with the asterisk (*). Control is automatically returned to the **Command Processor** if either the Execute Switch (/G) or the Exit Switch (/E) is included in the Command Line or if RESET or CTRL-C is depressed.

The Command Line contains the names of one or more files to be linked. This may be followed by a number of option switches. Each filename and option switch may be entered separately, prompted by the (*).

In the general form of the Command Line, the following abbreviations are used:

drv:	is the disk drive to be used; if omitted, the default is the logged-in drive.
filnam.REL	is the relocatable module (.REL) produced by the Assembler or Fortran Compiler (see ADDS*FORTRAN™ Reference Manual).
swl-swn	are the option switches which are described on page 7-2.

The general form of the Command Line is shown below:

```
*[drv:]filnaml[.REL],[drv:]filnam2[.REL],...,
                [drv:]filnamn[.REL]/swl/.../swn
```

OPTION SWITCHES

/D Form: /D:address,[drv:]filename[.REL]

The "Data Origin" or D Switch sets the data origin for the next program loaded. All data and COMMON areas of the specified filename are loaded starting at the data origin. The program is loaded starting at the program origin specified by the P Switch. If the P Switch is omitted, the program origin default is 1103H. If the data origin is less than the program origin, the user must be sure there is enough room to prevent the program from being destroyed.

/E Form: /E

The "Exit" or E Switch also searches FORLIB.REL to satisfy unresolved globals and then exits the Linking Loader and returns control to ADOS. The E Switch functions as an implied FORLIB/S. This switch is useful when loading a program and saving the memory image, as it displays the number of blocks of the temporary program area to be saved by the SAVE command as well as the starting address and the ending address of the program in memory (also see N Switch).

/G Form: /G

The "Execute" or G Switch starts execution of a program after searching FORLIB.REL to satisfy unresolved globals. The file FORLIB.REL can be included in the original command line to be searched before execution. Thus, the G Switch functions as an implied FORLIB/S (see S Switch). The G Switch will not save the executable module. However, before execution actually begins, the Linking Loader displays the following information:

```
[start          address of next          number of 256- ]
[address        available byte           byte pages used]

[BEGIN EXECUTION]
```

After execution is complete, the executable module may be SAVEd by using the SAVE command (also see N Switch and E Switch).

/H Form: /H

The "Hexadecimal" or H Switch prints the data and program origin addresses and the starting and ending addresses in hexadecimal. If neither H or O is specified, the system defaults to hexadecimal.

/M Form: /M

The "Map" or M Switch causes all global labels and their addresses to be listed. An asterisk next to a label indicates that the label is unresolved (i.e., the module containing the definition for the label has not been loaded). In addition, the origin and end of the data area are also printed. If a D Switch had been entered prior to the M Switch, the origin and end of the program area will also be printed. If not, the program is stored in the data area.

/N Form: [drv:]filename/N

The "SAVE" or N Switch saves the executable module in memory under the specified filename with a default filetype of COM after a /E or /G is entered. If necessary, a jump to the start of the program (at 1100H) is inserted.

/O Form: /O

The "Octal" or O Switch prints the data and program origin addresses and the starting and ending addresses in octal. The default is hexadecimal.

/P Form: /P:address,[drv:]filename[.REL]

The "Program Origin" or P Switch sets the program origin for the next program loaded. The program area of the specified filename is loaded starting at the program origin. If the D Switch is omitted the data and COMMON areas are also loaded at the program origin.

/R Form: /R

The "Reset" or R Switch aborts the link editor and is to be used whenever the wrong file is loaded. The internal pointer is moved back to the beginning and the Linking Loader assumes that no files are loaded. A new Command Line must then be entered.

/S Form: /S

The "Search" or S Switch causes a search for the specified (Library) filename .REL for unresolved globals, and loads portions of the file that satisfy those unresolved globals. This command should be used only after all desired modules have been loaded.

/U Form: /U

The "Undefined" or U Switch causes a display of all globals which are still undefined in all current modules and prints the origin and end of the data area. If a D Switch had been entered prior to the U Switch, the origin and end of the program area will also be printed. If not, the program is stored in the data area.

EXAMPLES OF LINKING LOADER COMMAND LINES

Example 1

a) *MYFIL1.REL,MYFIL2.REL,...,
MYFILn.REL,MYLIB.REL/S/E

b) *MYFIL1,MYFIL2,...,MYFILn,MYLIB/S/E

In example 1 above, versions a and b both load the following relocatable program modules into memory from the logged-in drive.

MYFIL1,MYFIL2,...,MYFILn,MYLIB

Version b is an abbreviated version of a with the filetypes assumed to be REL.

The "S" Switch searches MYLIB.REL to satisfy unresolved globals; the "E" Switch exits the Linking Loader after displaying, in hexadecimal, the starting address (11A1), ending address (2F82) and number of 256-byte pages of the temporary program area used (31) in the following form:

[11A1 2F82 31]

The SAVE command may be used to save the 31 blocks containing the executable module as follows:

A>SAVE 31 XFIL.COM

The executable file must have filetype COM. Once SAVEd, this absolute machine code file can be loaded and executed by entering the filename as follows:

A>XFIL

Relocatable program modules may be loaded from different disk drives. The Command Lines below load one module from drive A and two from drive B.

*A:MYFIL1.REL,B:MYFIL2.REL,B:MYFIL3.REL or

*A:MYFIL1,B:MYFIL2,B:MYFIL3

Example 2

```
a) *PGM1.REL,PGM2.REL,...,PGMn.REL/U/G

b) *PGM1,PGM2,...,PGMn/U/G
```

In example 2 above, a and b are two versions of the same Command Line. They both load the following relocatable program modules into memory from the logged-in drive.

```
PGM1,PGM2,...,PGMn
```

Version b is an abbreviated version of a with the filetypes assumed to be REL.

The "U" Switch displays all globals which are undefined in all current modules as well as the origin and end of the program and data areas in hexadecimal as shown below:

```
DATA 1103  3018
```

The "G" option satisfies all unresolved globals and begins execution of the first relocatable program stored in memory after printing, in hexadecimal, the starting address (11F7), ending address (3018) and number of 256-byte pages of the temporary program area used (32) and the BEGIN EXECUTION message as follows:

```
[11F7  3018  32]
[BEGIN EXECUTION]
```

The executable module may be SAVEd after execution by entering the following command:

```
A>SAVE 32 PGM.COM
```

If the O Switch is entered before the names of the relocatable files, all addresses are printed in octal instead of hexadecimal.

```
*/O,B:PGM1/U,G

DATA 010403  030030

[010767   030030   32]
[BEGIN EXECUTION]
```

Example 3

```
*B:PGM1
*TEST/N
*/G
```

The N Switch above saves the executable module as TEST.COM on drive A, after program execution initiated by the G Switch.

Example 4

```
*/D:5000/P:1200,B:PGM1

*B:PGM1/N/G
```

The D and P Switches above set the data origin at 5000H and the program origin at 1200H and display the following information:

```
DATA      5000    50F3

PROGRAM   1200    13C8
```

The N Switch saves the executable module as PGM1.COM on drive B after program execution.

If the P Switch had been omitted the program origin would default to 1103H.

```
*/D:5000,B:PGM1

DATA      5000    50F4

PROGRAM   1103    12CB
```

If the D Switch had been omitted, the program origin would be used as both a program and data area.

```
*/P:1200,B:PGM1

DATA      1200    14BC
```

ADDRESS CHAINING

Each time the Linking Loader encounters a reference to a symbol that has not yet been defined, it enters the address of the reference into a chain. Each entry in the chain contains a pointer to the previous entry. The last entry contains an absolute zero. When the symbol is defined, the Linking Loader goes through the chain again from the last entry to the first, replacing the contents of each entry with the assigned address of the symbol.

External references are handled by saving the unresolved chains from all of the modules. The contents of the first entry in a chain for one module is the address of the top of the chain for the previously loaded module.

At the end of each load or execute action, the Linking Loader automatically displays a list of symbols which remain undefined. The /U Switch can be used to display undefined symbols at any time before invoking /G.

LINKING LOADER ERROR MESSAGES

The Linking Loader detects two kinds of errors, warnings and fatal errors. Warning messages are preceded by percent (%) signs and fatal errors by question marks (?).

Fatal Errors

?Can't Save Object File	A disk error occurred when the file was being saved.
?Command Error	Unrecognizable L80 Command.
?<file> Not Found	A file in the command string did not exist.
?Intersecting [Program] Area [Data]	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.
?Loading Error	The last file given for input was not a properly formatted L80 object file.
?No Start Address	A /G Switch was issued, but no main program had been loaded.
?Origin [Above] Loader Memory, Move Anyway (Y or N)? [Below]	After an /E or /G is entered, 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 Y is entered, the Linking Loader will move the area and continue. If N or any other character is entered, the Linking Loader will exit. If /N had been entered previously, the image will already have been saved.

?Out of Memory	Not enough memory to load program.

Warning Messages

%2nd COMMON Larger /XXXXXX/	The first definition of COMMON block /XXXXXX/ was not the largest definition. Re-order 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 [Program] Area [Data]	A /D or /P will cause already loaded data to be destroyed.

SECTION 8:

DEBUG UTILITY

The Debug Utility allows dynamic interactive testing and debugging of assembly language programs generated under ADOS.

GENERAL COMMAND STRUCTURE

The Debug Utility is initiated by typing one of the following commands at the keyboard when the **Command Processor** is in control.

```
A>DEBUG <ENTER>
A>DEBUG [drv:]filename. <ENTER>
A>DEBUG [drv:]filename.filetype <ENTER>
```

Filename is the name of a program to be loaded and tested. In all cases, Debug is brought into main memory in the place of the **Command Processor**. When a filename is part of the command, it is automatically loaded into memory. If a disk drive is omitted, the currently logged-in disk is assumed.

Upon initiation, the following sign-on message is displayed to notify the user that the Debug Utility is in control.

```
ADDS DEBUG VER m.n
```

In the above message, m.n represents the current version number. If a filename is part of the command, the following sign-on message is also included:

```
NEXT  PC
NNNN  PPPP
```

where

NNNN is the next address following the loaded program,

PPPP is the assumed program counter (11ØØH for COM files).

Following the sign-on message, Debug prompts the operator with a hyphen character (-) and waits for input from the keyboard. The operator can type any of several single-character commands, terminated by the ENTER key to execute the command. Each line of input can be edited using the System 50/70 function keys DEL CHAR, INS CHAR, HOME, ERASE, left and right cursor control keys (see Section 2, System 50/70 Facilities).

Any command can be up to 32 characters in length, where the first character determines the command type. The command character may be followed by up to three hexadecimal values separated by commas or single blank characters. All numeric output from Debug is in hexadecimal form. In all cases, the commands are not executed until the ENTER key is depressed.

At any point in a debugging session, the operator can stop execution of Debug. One depression of the RESET key in the G mode returns the operator to the Debug Utility; a second depression of this key returns the operator to the **Command Processor**. After the CP has control, the operator can preserve the current memory image using the SAVE command.

In addition, anything displayed on the screen may be printed on an attached line printer by depressing the PRINT LOCAL key.

COMMANDS

This section describes the commands in detail. In each case, the operator must wait for the prompt character (-) before entering the command. In the explanation of each command, the command letter may be followed by numbers separated by commas. These numbers, represented by phrases, are always assumed to be hexadecimal numbers from one to four digits in length. Longer numbers are automatically truncated on the right.

Many of the commands operate on 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 (PC) and the stack pointer (SP) which default to 11ØØH.

1. ASSEMBLE Form: Astartaddr

In-line assembly language can be inserted into the current memory image using the A command, where the number following is the hexadecimal starting address for the in-line assembly. Debug prompts the operator with the address of the next instruction to fill, and reads the keyboard looking for assembly language mnemonics, followed by register references and operands in absolute hexadecimal form. Each successive load address is printed before reading the keyboard. The A command terminates when the first empty line is input.

Upon completion of assembly language input, the operator can review the memory segment using the disassembler (the L command).

The user should make full use of the INTEL manual entitled 8080/8085 Assembly Language Programming Manual 98-301A.

2. DISPLAY Form: D
Dstartadd
Dstartadd,endadd

The D command allows the operator to view the contents of memory in hexadecimal and ASCII formats. In the first case, memory is displayed from the current 192 locations (CØH) and continues for 12 display lines. Each display takes the following form:

```
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,

BB represents the data present in memory, starting at AAAA, and

CC.. are the ASCII characters starting at AAAA. A period is substituted for non-graphic symbols.

NOTE: Both upper and lower case alphabetics are displayed. 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 (Dstartadd) is similar to the first, except that the display address is first set to startadd, and 12 display lines are sent to the screen (192 locations = CØH).

The third form of the D command (Dstartadd,endadd) causes the display to continue from startadd through endadd. 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 may be aborted by depressing any key except CTRL-C. Depressing CTRL-S causes the display to halt; to continue the display, depress any key except CTRL-C.

3. FILL Form: Fstartadd,endadd,constant

The F command must have the starting address, ending address and a hexadecimal byte constant as arguments. Debug stores the constant at the starting address, increments the value of the starting address, and tests against the ending address. If the starting address is now greater than the ending address, then the operation terminates; otherwise, the operation is repeated. Thus, the F command can be used to set a memory block to a specific constant value.

4. GO Form: G
Gstartadd
Gstartadd,brkpt
Gstartadd,brkptl,brkpt2
G,brkpt
G,brkptl,brkpt2

The user enters the letter G to start program execution, with up to two optional breakpoint addresses.

G starts execution of the program under test at the current value of the program counter, in the current machine state, with no breakpoints set.

Gstartadd is similar to G except that the program counter in the current machine state is set to startadd before execution begins.

Gstartadd,brkpt is the same as Gstartadd except that program execution stops when the breakpoint address is encountered. The breakpoint address must be in the area of the program under test. The instruction at the breakpoint address is not executed when the breakpoint is encountered.

Gstartadd,brkptl,brkpt2 causes two breakpoints to be specified. Encountering either breakpoint causes execution to stop, and both breakpoints are then cleared.

G,brkpt and G,brkptl,brkpt2 take the program counter from the current machine state and set one and two breakpoints respectively. (The comma is required.)

Execution continues from the starting address in real-time to the next breakpoint. There is no intervention between the starting address and the breakpoint address by the Debug Utility. If the program under test does not reach a breakpoint, control can be returned to Debug by depressing RESET once.

Upon encountering a breakpoint, Debug stops execution and types *D which is the stop address. The machine state can be examined at this point using the X (examine) command.

The operator must specify breakpoints which are different from the program counter address at the beginning of the G command.

NOTE

Depressing RESET once during a G command causes a return to Debug. If RESET is depressed while in Debug, a return to ADOS will result.

5. INPUT Form: Ifilename
Ifilename.filetype

The I command allows the operator to insert a filename into the default file control block at 1Ø5CH. The default file control block can be used by the program under test as if it had been passed by the console **Command Processor**. This filename is also used by Debug for reading additional COM files.

If the second form is used and the filetype is COM, a subsequent R command can be used to read the pure binary machine code (see R below).

6. LIST Form: L
Lstartadd
Lstartadd,endadd

The L command is used to list assembly language mnemonics in a particular region of a program on the display.

L lists 11 lines of disassembled machine code from the current list address.

Lstartadd first sets the starting address and then lists 11 lines of code.

Lstartadd,endadd lists 11 lines of code beginning at the specified starting address and ending at the specified ending address.

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). Long typeouts can be aborted by depressing any key or CTRL-S for a temporary halt.

7. MOVE Form: Mstartadd,endadd,destadd

The M command allows block movement of program or data areas from one location to another in memory. The start of the move is denoted by the first argument, startadd; the ending address by the second argument, endadd; the destination address is given in the third argument, destadd.

Data is first moved from the starting address to the destination address and both addresses are incremented. If the starting address is greater than the ending address, the move operation stops; otherwise, the move operation is repeated.

8. READ Form: R
Rbiasadd

The R command is used in conjunction with the I command to read COM files from the diskette in preparation for the debugging session.

In the command form Rbiasadd, the argument 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 1ØØØH through 1ØFFH (the first page of memory). If the bias address is omitted, then a bias address=ØØØØ is assumed.

The R command requires a previous I command specifying the name of a COM file. An assumed load address of 11ØØH is taken for COM files. Any number of R commands can be issued following the I command to reread the program under test, assuming the tested program does not destroy the default area at 1Ø5CH. Moreover, any file specified with the filetype COM is assumed to contain machine code in pure binary form, created with the SAVE command.

Whenever the R command is issued, Debug responds with either the error indicator "?" or with a load message in the form

```
NEXT PC
NNNN PPPP
```

where

NNNN is the next address following the loaded program,
PPPP is the assumed program counter (11ØØH for COM files).

9. SET Form: Sstartadd

The S command allows memory locations to be examined and optionally altered, where the argument is the hexadecimal

starting address for examination and alteration of memory. Debug responds with a numeric prompt, giving the memory location along with the data currently held in the memory location. If the operator enters the space bar to increment the address, the data is not altered. If a byte value is typed, the value is stored at the prompted address. In either case, the prompts continue with successive addresses and values until either a period is typed by the operator or an invalid input value is detected.

10. TRACE Form: T
Tnumsteps

The T command allows selective tracing of program execution for 1 to 65535 program steps. In the first form (T) the CPU state is displayed and the next program step is executed. The program terminates immediately with the termination address displayed as *.... which is the next address to execute.

The display address used in the D command is set to the value of the H and L registers. The CPU state at program termination can then be examined using the X command.

The second form of the command is similar, except that execution is traced for numsteps, a hexadecimal value, before a program breakpoint is encountered. A breakpoint can be forced in the trace mode by depressing the RESET key. 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 below.

Program tracing is discontinued at the interface to ADOS and resumes after return to the program under test. Thus, ADOS 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 Debug 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 accomplish the break using RST7. Further, the trace mode always runs the tested program with interrupts enabled, which may cause problems if asynchronous interrupts are received during tracing.

11. UNTRACE Form: U

The U command is identical to T except that intermediate program steps are not displayed. The untrace mode allows from 1 to 65535 (ØFFFFH) 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. EXAMINE

Form: X
XREG

The X command allows selective display and alteration of the current CPU state for the program under test. In the second form of the command, the argument is one of the 8080 CPU registers or flags, as shown:

C	CARRY FLAG	Ø or 1
Z	ZERO FLAG	Ø or 1
M	MINUS FLAG	Ø or 1
E	EVEN PARITY FLAG	Ø or 1
I	INTERDIGIT CARRY	Ø or 1
A	ACCUMULATOR	Ø-FF
B	BC REGISTER PAIR	Ø-FFFF
D	DE REGISTER PAIR	Ø-FFFF
H	HL REGISTER PAIR	Ø-FFFF
S	STACK POINTER	Ø-FFFF
P	PROGRAM COUNTER	Ø-FFFF

When no register is given, the CPU register state is displayed in the following format:

```
CFZFMFEFIF A=BB B=DDDD H=DDDD S=DDDD P=DDDD INST
```

where

F	is a flag value = Ø or 1
BB	is a byte value
DDDD	is a double byte quantity corresponding to the register pair
INST	is the disassembled instruction which occurs at the location addressed by the CPU state's program counter.

In the second form of the command, the argument is one of the registers or flags given above (C,Z,M,E,I,A,B,D,H,S, or P) allowing the display and optional alteration of register values.

In each case, the flag or register value is first displayed on the screen. Debug then accepts input from the keyboard. If the space bar is depressed, 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.

NOTE

If an I or R command loads in a program that overwrites the first part of Debug (allowed), then the A and L commands are lost, as is the mnemonic display part of the X command.

APPENDIX A:

TEST MODE MESSAGES

Status Field Error Message	Description	Procedure
SB/VG FAIL	Failure in the Screen Buffer or Video Generator.	Unrecoverable: notify Supervisor and customer service.
M1 FAIL	Failure in Memory Card #1.	Unrecoverable: notify Supervisor and customer service.
M2 FAIL	Failure in Memory Card #2.	Unrecoverable: notify Supervisor and customer service.
M3 FAIL	Failure in Memory Card #3.	Unrecoverable: notify Supervisor and customer service.
PB FAIL	Failure in the Printer Buffer.	Depress the NEXT PAGE key and continue with all system functions except printer operations.
FB FAIL†	Failure in front-end Card B.	Depress the NEXT PAGE key and continue with all system functions except communications.
FA FAIL†	Failure in front-end Card A.	Depress the NEXT PAGE key and continue with all system functions except communications.
FC FAIL†	Failure in front-end Card C.	Depress the NEXT PAGE key and continue with all system functions except communications.
FM FAIL†	Failure in front-end memory.	Depress the NEXT PAGE key and continue with all system functions except communications.
FE TMOUT†	Unknown front-end error.	Depress the NEXT PAGE key and continue with all system functions except communications.

†System 70 only

Status Field Error Message	Description	Procedure
SYS/FAIL	Default failure (none of the above conditions).	Unrecoverable: notify Supervisor and customer service.
DB FAIL	Disk Controller	Unrecoverable: notify Supervisor and customer service.

COMMENT SHEET

LET US KNOW... if you have any recommendations for improving this publication. After writing your comments, fold, tape and mail this pre-addressed form to us. All comments will be given careful consideration, and become the property of Applied Digital Data Systems Inc.

(from the title page)
PUBLICATION TITLE _____ DATE _____

PUBLICATION NO. _____

YOUR NAME _____ TITLE _____

COMPANY _____

ADDRESS _____

_____

_____

REMARKS:

BUSINESS REPLY MAIL
NO POSTAGE STAMP NECESSARY IF MAILED IN THE UNITED STATES

POSTAGE WILL BE PAID BY:

FIRST CLASS
PERMIT NO. 105
SMITHTOWN
N.Y. 11787

ADDS

Applied Digital Data Systems Inc.
100 Marcus Boulevard
Hauppauge, New York 11787

ATTN: TECH PUBLICATIONS