

2300 ADS SOFTWARE
REFERENCE MANUAL
Manual No. 2300-5000-01

5730 Buckingham Parkway
Culver City, CA 90230

Copyright 1980

The 2300-ADS Software Reference Manual is divided into six sections as listed below.

Section 1 contains a summary of each ADS system program, a description of command and filename conventions, precautions and start-up information.

Sections 2 through 6 explain the purpose, operation and available commands for the ADS system programs: Manager, Editor, Assembler, Linker and Command File Processor.

Since operating procedures and commands vary between processors, the Debugger program is not described here. Refer to the appropriate personality manual for detailed debugging instructions.

Information regarding the Prolog PROM Programmer 920 and 900B units and the DATAIO PROM Programming unit is provided in Appendices A and B. Appendix C provides information on baud rates, parity values, character length and stop bits used with the serial ports. Appendix D defines ASCII character code conversion.

SECTION 1 - INTRODUCTION		Page
1.1	Software Development.....	1-1
1.2	Hardware Development.....	1-2
1.3	Keyboard.....	1-2
1.4	Command Conventions.....	1-2
	Commands.....	1-2
	Parameters.....	1-2
	Special Keys.....	1-4
	User Verification.....	1-4
	Command Completion.....	1-5
1.5	Filename Conventions.....	1-5
	Exceptions.....	1-6
1.6	Filename Prefixes.....	1-6
1.7	Filename Parameters.....	1-8
1.8	Diskettes.....	1-9
	Insertion Into Drive.....	1-9
	Physical Write Protection.....	1-10
1.9	Starting Procedure.....	1-11
1.10	Invoke Programs.....	1-11
1.11	Power Down.....	1-11
SECTION 2 - MANAGER		
2.1	Introduction.....	2-1
2.2	File Attributes.....	2-1
2.3	Diskette Initialization.....	2-2
2.4	Diskette Directory.....	2-2
2.5	Command Line Editing.....	2-3
2.6	Wild Card Operations.....	2-3
	Single Character Wild Card Symbol.....	2-4
	Previous Specification Wild Card Symbol.....	2-4
	Single, Multiple or No Character Wild Card Symbol.....	2-5
	Wild Carding by Attribute.....	2-5
	Manager Queries During Wild Card Operations....	2-6
2.7	Option Switches.....	2-6
	/A Switch.....	2-6
	/C Switch.....	2-6
	/Q Switch.....	2-6
	/P Switch.....	2-6
	/V Switch.....	2-7
	List of Commands.....	2-9
	List of Messages.....	2-19

SECTION 3 - EDITOR

3.1	Introduction.....	3-1
3.2	Editor Display.....	3-1
3.3	Editing Modes.....	3-2
3.4	Input and Output.....	3-2
	List of Commands.....	3-3
	List of Messages.....	3-19

SECTION 4 - ASSEMBLER

4.1	Introduction.....	4-1
	Assembler Input.....	4-1
	Assembler Output.....	4-2
4.2	Program Segments.....	4-4
4.3	Location Counters.....	4-4
4.4	Assembler Options.....	4-4
4.5	Specifying Assembler Files.....	4-5
4.6	Assembler Processing.....	4-6
4.7	Halt Assembly.....	4-6
4.8	Assembler Statement Syntax.....	4-6
	Label Field.....	4-6
	Operation Code Field.....	4-7
	Operand Field.....	4-7
	Comment Field.....	4-12
	Comment Statement.....	4-12
4.9	Assembler Directives.....	4-12
	Segment Directives.....	4-12
	Segment Origin Directive.....	4-13
	Global Directive.....	4-13
	END Directive.....	4-13
	EQU Directive.....	4-14
	Define Constants Directive.....	4-14
	Define Memory Space Directive.....	4-15
	Printer Control Directives.....	4-15
4.10	Macros.....	4-16
	Syntax.....	4-17
	Substitution.....	4-18
	Liberal Ampersand.....	4-18
	Placement of Macro Definitions.....	4-18
	Duplicate Macro Definitions.....	4-19
	Generating Unique Labels.....	4-19
	Concatenating Parameters.....	4-20
	Macro Calls Within Macros.....	4-20
	EXITM Statement.....	4-20
4.11	Conditional Assembly	4-21
	DEFL Statement.....	4-21
	DEFG Statement.....	4-21
	SUBSTR Statement.....	4-22
	LENGTH Statement.....	4-22

	IF Block.....	4-22
	DO Block.....	4-24
4.12	Advanced Features.....	4-25
	Subscripted Set Symbols.....	4-25
	Indirect Set Symbols.....	4-26
	List of Assembler Messages.....	4-27
	List of Macro and Conditional Assembly Messages.....	4-29

SECTION 5 - LINKER

5.1	Introduction.....	5-1
5.2	Linker Input.....	5-1
5.3	Linker Output.....	5-1
5.4	Options.....	5-1
5.5	Commands.....	5-2
5.6	Memory Map Output.....	5-5
5.7	Reference List Output.....	5-6
	List of Messages.....	5-7

SECTION 6 - COMMAND FILE PROCESSOR

6.1	Introduction.....	6-1
6.2	Command File Creation.....	6-1
6.3	Initiate Command File Processing.....	6-1
6.4	Command File Processor Features.....	6-2
	^L Feature.....	6-2
	^K Feature.....	6-3
	^n Feature.....	6-5
	^^ Feature.....	6-5
6.5	Keyboard-Only Input.....	6-6
6.6	Errors.....	6-6
6.7	Abort Processing.....	6-6
	List of Messages.....	6-7

APPENDIX A - PROLOG PROM PROGRAMMER INTERFACES

APPENDIX B - DATAIO PROM PROGRAMMER INTERFACE

APPENDIX C - SERIAL PORT VALUES

APPENDIX D - AMERICAN STANDARD CODE FOR INFORMATION INTERCHANGE (ASCII)

LIST OF ILLUSTRATION

Figure		Page
1-1	2300 ADS Keyboard.....	1-3
1-2	Insertion Into Drive.....	1-10
1-3	Physical Write Protection.....	1-10
3-1	Editor Display.....	3-2
4-1	Example Assembler Input.....	4-2
4-2	Example Assembler Output.....	4-3
4-3	Example Assembler Reference List.....	4-3
4-4	Example Assembler Display.....	4-5
4-5	MACRO Definition.....	4-17
5-1	Example Linker Display.....	5-3
5-2	Example Memory Map Output.....	5-5
5-3	Example Linker Reference List Display.....	5-6

LIST OF TABLES

Table		Page
2-A	File Attributes.....	2-1
3-A	Keyboard Editing Operations.....	3-17
4-A	Assembler Options.....	4-4
5-A	Linker Options.....	5-2
A-1	920 8080/8085 Interface Diskette.....	A-1
A-2	920 Z80 Interface Diskette.....	A-2
A-3	900B 8080/8085 Interface Diskette.....	A-3
A-4	900B Z80 Interface Diskette.....	A-4
A-5	Pin Connections.....	A-5
A-6	Jumper Connections.....	A-6
B-1	DATAIO 8080/8085 Interface Diskette.....	B-1
B-2	DATAIO Z80 Interface Diskette.....	B-2
B-3	Pin Connections.....	B-3
B-4	Jumper Connections.....	B-3
D-1	ASCII Character Code Conversion.....	D-1

1.1 SOFTWARE DEVELOPMENT

The 2300-ADS provides five major programs for generating and perfecting user software.

- 1) Manager. The Manager provides an organization system for storing information on diskette. Data are grouped into "files", and the name, type and length of each file are recorded in a diskette directory which may be displayed or printed. As files are added, modified or deleted, the Manager automatically updates the directory.
- 2) Editor. Programs or text may be entered from the keyboard or loaded from a diskette file into the Editor for modification. Any size file may be edited, and more than 40 thousand characters of text may be loaded into the Editor's work space at one time. The scrolling (arrow) keys are used to position work space information for viewing on the display screen.
- 3) Assembler. The Assembler translates assembly language programs into relocatable object code and stores it in diskette files. During translation, the Assembler detects syntax errors and displays or prints the incorrect lines.
- 4) Linker. The Linker selects relocatable program segments (RSEGs) from relocatable object files, links and locates them at absolute addresses. The result is then written to produce one executable absolute object file.
- 5) Debugger. The Debugger allows a program to be executed and examined for errors. Debugging is performed either with the 2302 Slave Emulator or with plug-in system modules. Since features included in the Debugger facility vary from processor to processor, the Debugger is not described here. Refer to the appropriate personality manual for specific operating instructions.

A Command File Processor feature is also provided. This processor reads command lines from a diskette file and passes them to ADS programs, thus providing a convenient alternative to the time-consuming method of keyboard command entry. The Command File Processor is particularly helpful when long sequences of commands must be entered repeatedly.

INTRODUCTION

1.2 HARDWARE DEVELOPMENT

Two separate hardware facilities are available for hardware development: the 2300-ADS Emulator for several 8-bit processors and the 2302 Slave Emulator for the newest 8-bit processors and for 16-bit processors. Refer to the 2300-ADS Emulator Manual or the 2302 Slave Emulator Manual for hardware development instructions.

1.3 KEYBOARD

Refer to Figure 1-1. The special keys on the left side of the keyboard are used for text editing or Command Line editing. The arrow or "scrolling" keys on the right side are used to position memory on the display screen. The **BREAK** and **STEP** keys, located on the far right, control hardware and Debugger functions as well as wild card operations in the Manager. The **LOAD** key enters the bootstrap loader. The **RESET** key re-initializes ADS system programs. All other keyboard characters are used for entering programs, commands and text. The functions of all special keys are described in Section 2.5 and the List of Commands in Section 3.

1.4 COMMAND CONVENTIONS

The conventions described here apply to all 2300-ADS system programs, unless otherwise specified.

Commands.

In this manual commands are shown in upper case, although they may be entered in either upper or lower case. The syntax descriptions show only the letters to be entered on the keyboard. For example, to initiate the Manager's Display command, the user enters:

D

Parameters.

For clarification purposes, parameters are shown in lower case, although they may be entered in either upper or lower case.

Optional parameters are enclosed in brackets:

[parameter]

Required parameters are enclosed in braces:

{parameter}

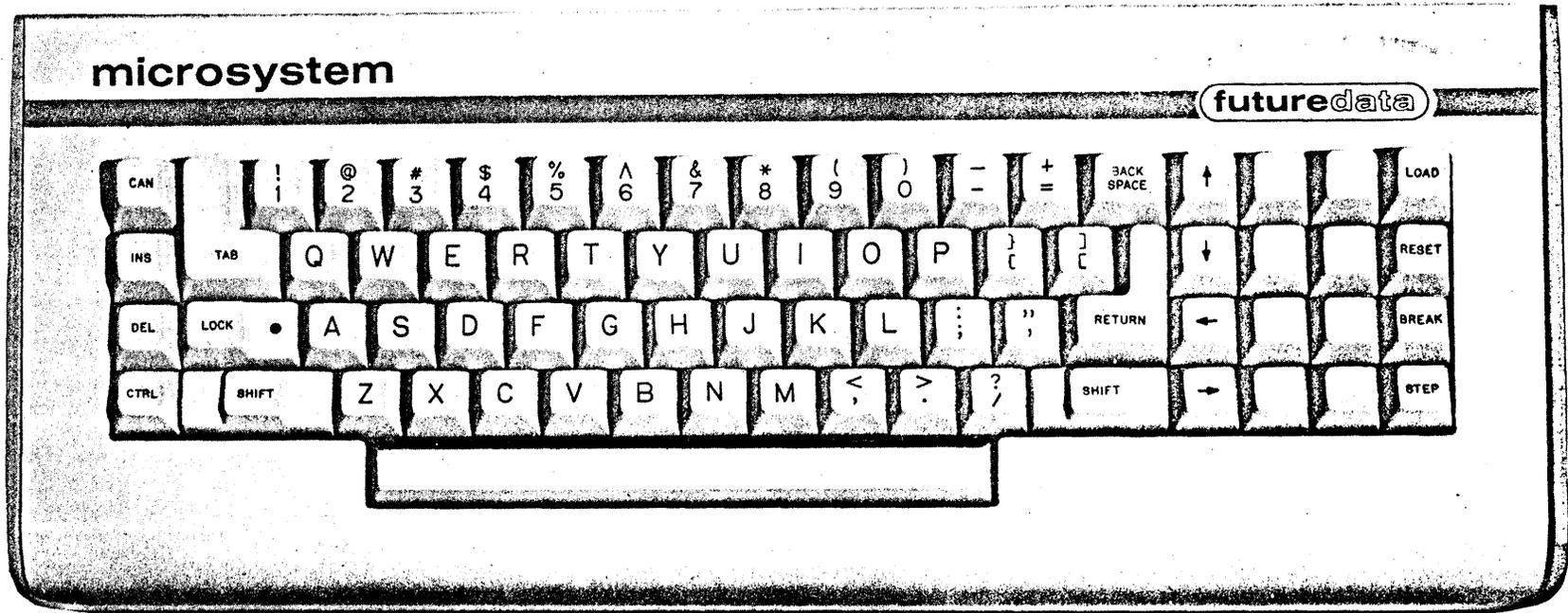


Figure 1-1. 2300 ADS Keyboard

INTRODUCTION

Brackets and braces are shown for clarification purposes only; they should not be entered with the parameters.

A series of horizontal or vertical dots in command syntax indicates that the parameter may continue to be entered indefinitely. For example, after the required expression is entered with the Editor's Find command, optional expressions may be entered:

F{expr}{[,expr]...

When one of several parameters may be selected, the available selections are listed vertically. In the example below, the user may enter the Editor's A command without parameters, with a decimal number (n) or with a dollar sign.

A $\left[\begin{array}{l} n \\ \$ \end{array} \right]$

Special Keys.

A letter, series of letters or symbol enclosed in a rectangle indicates that a special key should be used. For example:

$\boxed{\text{TAB}}$

indicates the tab key.

$\boxed{\text{DEL}}$

indicates the delete key.

User Verification.

Commands that prompt for verification (other than wild card operations) are executed upon entry of:

Y

for yes, or

N or $\boxed{\text{RETURN}}$

for no.

Commands that prompt for user verification during wild card operations are executed upon entry of

Y

for yes or

N, `RETURN`, `TAB`, or `STEP`

for no. `BREAK` aborts the operation.

Command Completion.

Unless otherwise specified, the `RETURN` key must be pressed at the end of a command line in order for the command to be executed.

1.5 FILENAME CONVENTIONS

In the syntax descriptions that follow, the word "filename" refers to a filename without prefixes or parameters. The word "file-spec" refers to a filename including any optional prefixes or parameters (see Sections 1.6 and 1.7 for information on prefixes and parameters).

ADS system programs accept only filenames conforming to the following conventions.

- 1) A filename must be from one to ten characters in length.
- 2) The first character must be a letter. Subsequent characters may be letters, digits or periods (see Exceptions on page 1-6).
- 3) Spaces are not allowed.

Several example filenames are listed below:

TESTPROG	- Valid
FILE29	- Valid
DRIVER 3.R	- Valid
BIG.PROGRAM	- Invalid; longer than ten characters

INTRODUCTION

- | | |
|----------|--|
| 9900PROG | - Invalid; first character is not a letter |
| MY-FILE | - Invalid; hyphen is an illegal character |

Exceptions.

Characters other than letters, numbers and periods may appear in a filename if the filename is enclosed in quotation marks. Also, if enclosed in quotes, a filename may begin with a non-alphabetic character. For example, the two invalid names, 9900PROG and MY-FILE can be validly specified as:

```
"9900PROG"  
"MY-FILE"
```

A filename may legally contain a quotation mark if it is followed by a second quotation mark and if the entire filename is enclosed in quotes. For example:

```
"ABC""DEF"
```

specifies file ABC"DEF.

1.6 FILENAME PREFIXES

Three types of prefixes may optionally precede a filename. Any two types or all three types may be used in the same filename specification, provided they are entered in the following order:

```
[new/old-spec:][device-spec:][(attribute-spec)]
```

- 1) New/Old Prefix. The ADS assumes that when an output filename is called for, the existing file should be overwritten or, if it does not exist, a new file should be created. In some instances the user may wish to override this default. Two prefixes are available for this purpose.
 - a) N: Prefix. The N: prefix instructs the ADS to create a new file, but not to write over the file if it exists. Examples:

N:OUTFILE	- specifies a new file named OUTFILE;
N:1:DEVTABLE	- specifies a new file on drive 1 named DEVTABLE.

- b) O: Prefix. The O: prefix instructs the ADS to write over an existing file and insures that an error will result if the file does not exist. Examples:

O:OUTFILE - specifies an existing file on drive 0
named OUTFILE;

O:1:(S)TEXT2 - specifies an existing source file on
drive 1 named TEXT2.

- 2) Device Prefix. A device prefix specifies the physical device to or from which a file will be transferred. The various disk drives, the printer and the serial output ports are treated as file-oriented devices by the ADS. Thus, the user may select any available device by entering a device prefix with a filename. If a device prefix is not entered, the ADS defaults to disk drive 0.

- a) Disk Drive. A disk drive is specified by prefixing the filename with the desired drive number followed by a colon. For example:

1:CMDPROG
0:TABLE.Z80

- b) Parallel Port Printer. Any output file may be transferred directly to the printer by entering:

P:

P: is the entire filename specification. For example:

M PROG,P:

transfers the file PROG to the printer. Refer to the List of Commands at the end of Section 2 for more information on the M command.

- c) Serial Port. A serial port is specified as:

S[n]:

where n is the optional port number. The default port number, as well as the baud rate and parity options may be preset using the Manager's I command (see List of Commands at the end of Section 2). Presettings may be overridden by including parameters in the filename specification (see Section 1.7).

INTRODUCTION

- 3) Attribute Prefix. The Manager program permits the user to specify the attributes associated with a file. (These specifications are one-letter codes which identify the file type. Refer to Section 2.2 for additional information on file attributes.) When it is necessary to specify an attribute(s) with a filename, enclose the attribute(s) in parentheses. For example:

(S)NEWFILE	- indicates a source file named NEWFILE;
(PO)MASTER	- indicates a permanent object file named MASTER;
1:(R)SUBR2	- indicates a relocatable file named SUBR2.

1.7 FILENAME PARAMETERS

In some instances, additional information (other than the filename itself and optional prefixes) is required for a filename specification. For example, when a serial port file is specified, parameters may be appended to specify the baud rate and parity options. A filename parameter is of the form:

{file-spec}[/parameter name=parameter value]

Two types of parameter values may be used.

- 1) Integer Parameter Value. An integer parameter value is used when a decimal number must be specified. For example, when a serial output port is specified, the baud rate may be specified as well:

I S2:/BAUD=1200

In the example above, serial port 2 is initialized to 1200 baud. Refer to Appendix C for additional information on baud rates.

- 2) String Parameter Values. String parameter values are similar to integer parameter values except that the value following the equal sign may be any string of characters. If characters other than letters, numbers or periods are included, the string must be enclosed in quotation marks. A quotation mark may appear in a string parameter if it is followed by a second quotation mark.

String parameters are necessary when specifying serial port parity. For example:

/PARITY=ODD

Refer to Appendix C for additional information on parity.

When multiple filename parameters are specified, each parameter is delimited by the slash that begins the parameter. Blanks may be entered between parameters as shown in the example below.

```
I S1: /LENGTH=7 /BAUD=600 /PARITY=EVEN
```

This command initializes serial port 1 to a character length of 7, a baud rate of 600, and even parity.

1.8 DISKETTES

Diskettes are coated with a magnetic material which is essential for storing information. A plastic jacket covers and protects most of the magnetic surface, but for additional protection and to insure diskette reliability, the following rules should be observed.

- 1) Never touch the magnetic surface.
- 2) When a diskette is not in use, it should be stored in the paper envelope provided.
- 3) Never bend a diskette.
- 4) Never subject a diskette to magnetic influences. Do not store near power transformers or motors.
- 5) Do not subject a diskette to temperatures below 50°F (10°C) or above 125°F (50°C).
- 6) Use only felt tip pen (never ball point pen or lead pencil) to write on a diskette label.
- 7) Do not expose a diskette to sunlight.
- 8) Do not attempt to clean the magnetic surface; abrasions may cause damage.

Insertion Into Drive.

Refer to Figure 1-2. To open the drive door, depress the drive door latch. Insert the diskette with the label to the right. When the diskette has been inserted completely, slide the drive door closed.

INTRODUCTION

Physical Write Protection.

Refer to Figure 1-3. A diskette with an uncovered write protection hole or notch can be accessed for reading, but not for writing. Stick-on tabs supplied with each package of diskettes may be used to "unprotect" the diskette and allow writing. If stick-on tabs are not available, opaque tape may be used to cover the write protection notch.

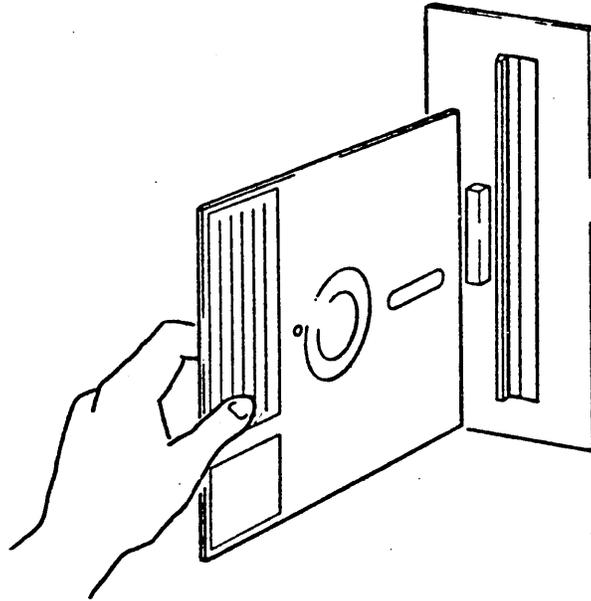


Figure 1-2. Insertion Into Drive.

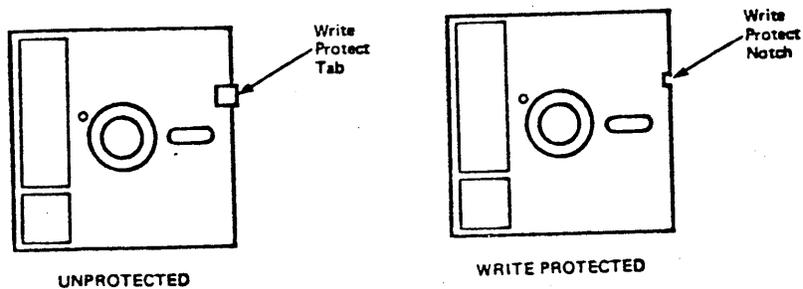


Figure 1-3. Physical Write Protection.

1.9 STARTING PROCEDURE

Follow the steps below to begin ADS operation.

- 1) Plug the disk drive unit into the AC connector on the ADS back panel.
- 2) Turn on the power switch on the ADS back panel.
- 3) Turn on the disk drive unit.
- 4) Insert the ADS system diskette into drive 0.

1.10 INVOKE PROGRAMS

According to the following list, type J{character} to load the desired ADS system program.

M = Manager
E = Editor
A = Assembler
L = Linker
D = Debugger

1.11 POWER DOWN

Diskette-stored information may be altered or erased if left in the disk drive unit(s) during power down. Remove all diskettes from all disk drives before turning off power.

2.1 INTRODUCTION

The Manager provides a convenient organization system for storing source files, relocatable object files, executable object files, text files, etc., on diskette. Groups of information (files) are given names by the user and stored on one or more of the 76 available tracks on each diskette. Files may be expanded, copied, deleted, etc., by entering the appropriate keyboard commands.

2.2 FILE ATTRIBUTES

File attributes help identify files and protect them against accidental overwriting or deletion. The available attributes are defined below:

- O executable absolute object file
- P permanent file
- R relocatable object file
- S source file
- W write-protected file
- Z permanent write-protected system file

ADS system programs require input file attributes and assign output file attributes as listed in Table 2-A.

Table 2-A. File Attributes

Program	Input File Attributes	Output File Attributes
Editor	S	S
Assembler	S	R and S
Linker	R	O and S
Debugger	O	O
Command File Processor	S	-

MANAGER

ADS system programs accept only files with the correct attribute. For example, a relocatable object file (R attribute) cannot be assembled because the Assembler accepts only source (S attribute) files. File attributes may be removed and/or changed to accommodate ADS system program requirements by use of the Manager's Attribute or Rename command (see List of Commands at the end of Section 2).

2.3 DISKETTE INITIALIZATION

ADS system diskettes have been configured in a 77-track format. All new or non-ADS system diskettes must be initialized for use in an ADS development system. Initialization defines the diskette boundaries for stored information and erases any previously stored data.

To perform the initialization process, type:

```
I {drive number} [/S=sector spacing][/C][/P]
```

A drive number must be specified. If the sector spacing parameter is omitted, the Manager defaults to 8 (6800/02 defaults to 12). For 4MHz CPUs, faster access will result in most situations if 4 is specified in the sector spacing parameter. As a safety precaution, the I command requests confirmation before beginning the initialization process. For information on the /C and /P option switches, refer to Section 2.7.

2.4 DISKETTE DIRECTORY

Upon initialization, the Manager creates a one-track diskette directory file (named DIR) which contains space for the filenames, file attributes and the number of tracks allocated to each file. The filenames in the directory may be displayed in alphabetical order by typing:

```
D[drive number]
```

If the drive number is omitted, the Manager defaults to drive 0. To print the directory, type:

```
D[drive number]/P
```

The Manager automatically updates the directory as files are added, modified or deleted. Refer to the List of Commands at the end of Section 2 for wild card operations with the D command.

NOTE 1

In the List of Commands at the end of Section 2, wild card parameters are designated by the words "wild-spec".

NOTE 2

Wild card specifications may be used only with Manager commands. The Editor, Assembler, Linker, Command File Processor and Debugger do not support wild card operations.

Single Character Wild Card Symbol.

The question mark (?) character within a filename implies any single legal filename character occurring in the same position in the name. For example:

M PROG?,1:PROG?

moves the files PROG1, PROG2 and PROGX from drive 0 to drive 1, but not the files PROG10, PROGRAM or PROG.

Previous Specification Wild Card Symbol.

The dollar sign (\$) may be used to specify a previous entry. For example, the command:

M PROG?,1:PROG?

can be simplified by using the \$ symbol:

M PROG?,1:\$

Both commands perform the same function.

Single, Multiple or No Character Wild Card Symbol.

The asterisk (*) character within a filename implies any group of zero or more characters. For example, using the Move command:

```
M*,1:*
```

moves all files (except those whose filenames are enclosed in quotes) from drive 0 to drive 1. The command:

```
M*,1:"*"
```

moves all files (including those whose filenames are enclosed in quotes) from drive 0 to drive 1.

To scratch (delete) a collection of files from drive 0 whose filenames begin with the characters SOURCE, the command:

```
S SOURCE*
```

is used. The files SOURCE.0, SOURCE.N and SOURCE123 are deleted, but the files RESOURCE and ASOURCE1 are not affected.

To assign attributes to a collection of files on drive 0 containing the string GO, the command:

```
R*GO*,(S)$
```

is used. Every file whose name contains the string GO (such as AGONY, GOSH or AGO) is assigned the S attribute.

Wild Carding by Attribute.

Files may be wild carded by attribute, as well as by name, or by a combination of the two. The attribute specification, shown in parentheses, must follow the drive number and colon, if present. For example:

```
M (S)*,1:*
```

moves all source files from drive 0 to drive 1.

If multiple attributes are coded, only those files having all of the specified attributes are processed. For example:

```
S (OW)*
```

scratches only those files with both the O and W attributes. Files with only one of these attributes are not scratched.

MANAGER

Manager Queries During Wild Card Operations.

When the Manager prompts for verification before executing a command, the user may respond with Y (the file is processed) or N, STEP, TAB or RETURN (the file is skipped). After receiving verification from the user, the Manager will continue to prompt for subsequent files. To abort the entire operation, the BREAK key is pressed.

When the Manager prompts for information such as attributes during an A command, the TAB key or STEP key may be used to skip the file and leave its attributes unchanged. Entering a RETURN clears all attributes from the file.

2.7. OPTION SWITCHES

Many Manager commands may be appended with an option switch. Five switches are available. Each switch must begin with a slash. Refer to the List of Commands at the end of Section 2 for specific uses of the option switches as they apply to individual commands.

/A Switch.

Some commands ask for user verification before executing the commanded function. The /A option switch may be used to override this verification. When the /A switch is specified, all files are processed as if Y were typed in response to each query.

/C Switch.

If the number of files processed exceeds the space available on the display screen, the Manager will pause and the message "PRESS ANY KEY TO CONTINUE" will appear. The /C option switch suppresses this message as well as all queries and prompts (the Manager assumes Y). The /C switch is particularly helpful when using command files and the length of displayed information is unknown.

/Q Switch.

The /Q switch requests verification for every case. Thus, the user may select a subset of the wild carded files by typing Y for yes, N, STEP, TAB or RETURN to skip to the next case, or BREAK to abort the operation.

/P Switch.

The /P switch routes all data on the display screen to the Centronix compatible printer.

/V Switch.

The /V switch may be used with a Move (M) command to compare the moved files to the original files.

ASSIGN ATTRIBUTES

A{file-spec} [/A=attribute] [/P]

A{wild-spec} [/A=attribute] [/P] $\left[\begin{array}{l} /A \\ /C \\ /Q \end{array} \right]$

PURPOSE

Attributes help identify files and protect them against accidental destruction.

PARAMETERS

/A assigns new attributes and overwrites any existing attributes. Any of the following attributes may be specified:

- O executable absolute object file
- P permanent file
- R relocatable object file
- S source file
- W write-protected file
- Z permanent, write-protected file

NOTES

A file may be assigned any combination of attributes. Attributes may be added and removed to accommodate ADS system program requirements (see Section 2.2, Table 2-A).

The wild card feature may be used to change the attributes of a group of files. If the attributes parameter is omitted, the Manager will prompt for the attributes for each file individually. The TAB key may be used to leave the attributes of any file unchanged and skip to the next file.

See Section 2.7 for option switch definitions.

MANAGER

LIST OF COMMANDS

CREATE FILE

C {filename} [/I=initial-alloc[/E=extension-alloc]] [/P] [/C]

PURPOSE

The C command creates a new file and allocates for that file the number of tracks specified in the /I parameter.

PARAMETERS

/I specifies the initial size of the file. When data are written into the file and the file exceeds its initial allocation, additional space is automatically allocated until the disk becomes full.

/E specifies the number of additional tracks to be allocated when the file expands beyond its allocated size. If this parameter is omitted, the default is one track at a time (each time the file expands beyond its allocated size).

NOTES

As stated in Section 1.6, new files are created automatically. The C command provides the additional feature of allowing a track allocation to be specified.

The C command does not support the wild card feature; only one file may be created at a time.

See Section 2.7 for option switch definitions.

DISPLAY

D{wild-spec}[/C][/P]

D[drive number][/C][/P]

PURPOSE

The D command displays the diskette directory (DIR) which shows the filenames (alphabetically sorted), file attributes, the number of tracks allocated to each file, and the number of remaining free tracks on the diskette.

NOTES

If a wild card filename or attribute specification is given, only the files matching the specification will be listed.

See Section 2.7 for option switch definitions.

EXCHANGE FILENAMES

E{filename one},{filename two}

E{wild-spec}[wild-spec] $\begin{bmatrix} /A \\ /C \\ /Q \end{bmatrix}$ [/P]

PURPOSE

The E command exchanges the specified filenames (this does not exchange filename prefixes).

NOTES

Only the names are exchanged; all other information remains the same.

See Section 2.7 for option switch definitions.

MANAGER

LIST OF COMMANDS

FREE UNUSED SPACE

F {filename}

F {wild-spec} $\left[\begin{array}{l} /C \\ /Q \end{array} \right] [/P]$

PURPOSE

The F command releases unused diskette tracks from the specified files.

NOTES

See Section 2.7 for option switch definitions.

INITIALIZE

The I command performs two functions: initializing a diskette and initializing a serial port. Each function is described here separately.

INITIALIZE DISKETTE

I{drive number}[/S=sector spacing][/P]

PURPOSE

The I command defines the diskette boundaries needed to access diskette-stored information, erases any previously stored data and creates a one-track directory file named DIR.

PARAMETERS

A drive number parameter must be specified.

/S may be an integer between 4 and 12, inclusive. If this parameter is omitted, the Manager defaults to 8 (6800/02 defaults to 12). For 4MHz CPUs, faster access will result in most situations if 4 is specified in the sector spacing parameter.

NOTES

As a safety precaution, the I command requests confirmation before beginning the initialization process. When initialization is complete, the diskette directory will be displayed.

See Section 2.7 for option switch definitions.

MANAGER

LIST OF COMMANDS

INITIALIZE SERIAL PORT

I{S[n]:}[/BAUD=n][/PARITY=letter][/LENGTH=n][/STOPS=n]

PURPOSE

The I command initializes and presets the serial port parameters. The ports must be initialized before they can be used.

PARAMETERS

If a port number (n) is not specified, serial port 1 is automatically set to the values specified in the /BAUD, /PARITY, /LENGTH and /STOPS parameters. The defaults for these parameters are:

/BAUD=300
/PARITY=none
/LENGTH=7
/STOPS=1 bit

NOTES

Once a serial port parameter value is set, it remains until it is reset or until the **LOAD** key is pressed.

Refer to Appendix C for the available values.

MOVE FILES

M{from filename},{[N:]to filename}{[/V]}

M{input wild-spec}{[,output wild-spec] $\begin{bmatrix} /A \\ /C \\ /Q \end{bmatrix}$ [/P][/V]}

M{input wild-spec},{[N:]to filename} $\begin{bmatrix} /A \\ /C \\ /Q \end{bmatrix}$ [/P][/V]}

PURPOSE

The M command moves (copies) the specified file into an existing file or a newly created file. The copied file remains unchanged in the "from" location.

PARAMETERS

The to filename parameter may contain any of the valid prefixes discussed in Section 1.6.

If the output wild-spec parameter is omitted, the Manager will prompt for each output name.

NOTES

See Section 2.7 for option switch definitions.

MANAGER

LIST OF COMMANDS

RENAME FILE

R{current file specification},{new file specification}

R{input wild-spec}[,output wild-spec] $\begin{bmatrix} /A \\ /C \\ /Q \end{bmatrix}$ [/P]

PURPOSE

The R command renames an existing file.

PARAMETERS

If the output wild-spec parameter is omitted, the Manager will prompt for each output name.

NOTES

See Section 2.7 for option switch definitions.

SCRATCH FILE

S {filename}

S {wild-spec} $\left[\begin{array}{l} /A \\ /C \\ /Q \end{array} \right] [/P]$

PURPOSE

The S command scratches (deletes) the specified file(s) from the diskette and frees the occupied track(s).

NOTES

As a safety precaution, the S command requests confirmation before beginning deletion.

See Section 2.7 for option switch definitions.

MANAGER

LIST OF COMMANDS

VERIFY

V{filename},{filename}

V{wild-spec},{wild-spec}

PURPOSE

The Verify command compares two files or groups of files for equality.

EXAMPLES

The following command compares file ABC on drive 2 to file XYZ on drive 2:

V2:ABC,2:XYZ

In the next example, the diskette in drive 0 is compared with the diskette in drive 1:

V*,1:"*"

LIST OF MESSAGES

BAD VERIFY

The V option was specified with an M command, and the moved information was not transferred correctly. Retry with another diskette.

DISK FULL

Diskette track space has been requested, but none is available. Use the F command to free unused space or use another diskette.

DRIVE NOT READY

The drive door is open, the drive has not yet attained operating speed, the diskette was inserted incorrectly, or the diskette is defective.

DUPLICATE NAME

The name specified for creating or renaming a file already exists on the specified diskette. Respecify the filename.

FILE NOT FOUND

The requested file does not exist, an erroneous drive number was specified, or the filename was typed incorrectly. Respecify the filename.

INVALID ATTRIBUTE

The Manager accepts only filenames with any of the following attributes: O, P, R, S, W or Z. Use the R command to change or remove the attributes.

INVALID NAME

The filename conventions were not followed or the filename prefixes and/or parameters were entered out of sequence. Respecify.

PARAM ERR

The requested ADS system program (J command) cannot be found on the diskette in drive 0 or an invalid command letter or parameter was entered in an ADS system command. Respecify.

PERM FILE

The file referenced for deletion has the permanent (P) or permanent write-protected (Z) attribute and cannot be deleted until the P or Z attribute has been removed. Use the R command.

MANAGER

LIST OF MESSAGES

PERM I/O ERR

A permanent diskette input or output error was detected. The cause may be one of the following: a worn or defective diskette, an attempt to write into or initialize a physically write-protected diskette, or an attempt to read or write to an uninitialized diskette.

PRESS ANY KEY TO CONTINUE

The amount of information to be displayed exceeds the length of the display screen. Press any key except **[BREAK]**, **[RESET]** or **[LOAD]** to display the remaining information.

SYNTAX

A command containing a syntax error was entered. Respecify.

WRITE PROTECT

A file having the write protect (W) attribute was selected for output, or the write-protect notch is uncovered. Remove the W attribute with the R command, or cover the write-protect notch.

3.1 INTRODUCTION

The Editor allows text to be created, corrected or expanded through use of several simple keyboard commands. Text is entered from the keyboard or loaded from a diskette file into the Editor's "work space", which can hold more than 40 thousand text characters. Large files can be read into the work space, edited and written out a portion at a time. Text in the work space may be viewed on the display screen up to 21 lines at a time.

The Editor's Text Block commands provide a time-saving alternative to the conventional line-by-line method of editing. An entire block of text can be moved, copied or deleted within the Editor's work space. When extensive editing is necessary, text blocks can be copied to an external file or merged into the work space at the desired location.

3.2 EDITOR DISPLAY

The Editor display is made up of four elements as described below (see Figure 3-1).

- 1) Edit Line. Most modifications to text must be entered on the Edit Line, which is located in the center of the display, surrounded by two horizontal dashed lines. The cursor may be moved to the Edit Line from the Command Line with the `RETURN` key. Text may be positioned for editing by use of the arrow keys or the Advance and Back-up commands.
- 2) Tab Line. The Tab Line is the horizontal dashed line located just above the Edit Line. Tab stops are indicated on the Tab Line by single vertical lines. The Editor automatically sets tab stops at character positions 11, 19 and 42, which is convenient for assembly language programs. Tab stop positions may be cleared and/or changed as desired.
- 3) Command Line. Upon entry to the Editor, the cursor appears on the Command Line, located at the bottom of the display. When the cursor is in this position, the Editor is ready to accept keyboard commands.
- 4) Message Line. The Editor displays error messages, queries and prompts on the line located just above the Command Line. The Message Line temporarily replaces the bottom line of displayed text.

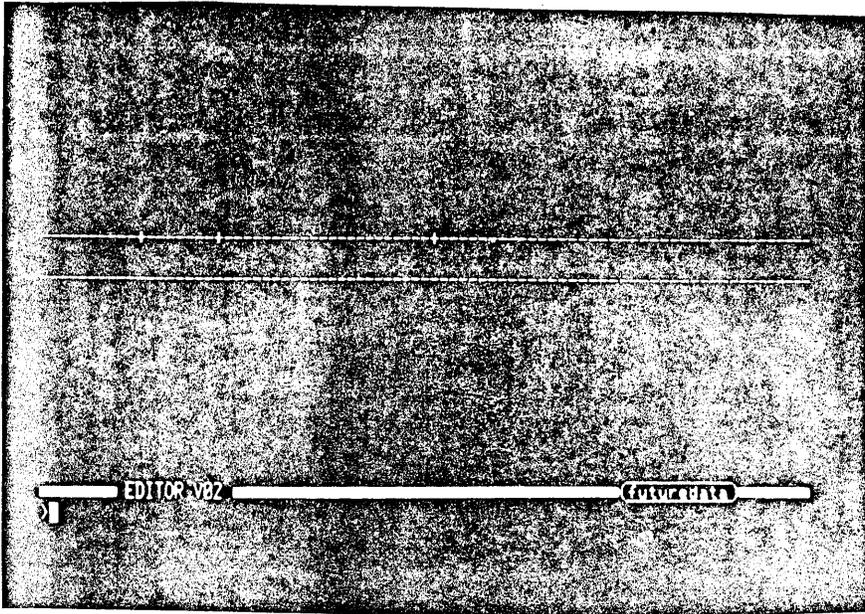


Figure 3-1. Editor Display.

3.3 EDITING MODES

The Editor provides three main modes as described below.

- 1) Command Mode. Any command (except special keys) may be entered to the Editor when in Command Mode. The Editor is in Command Mode when the cursor is located on the Command Line.
- 2) Line Edit Mode. The Editor enters Line Edit Mode when the `RETURN` key is pressed; `RETURN` moves the cursor to the Edit Line. Text on the Edit Line may then be modified as desired.
- 3) Line Insert Mode. The I command invokes Line Insert Mode which allows new text lines to be entered. Two successive depressions of `RETURN` cause the Editor to leave Line Insert Mode and the cursor to be moved to the Command Line.

3.4 INPUT AND OUTPUT

Diskette files loaded into the Editor must possess an S attribute (refer to the Manager's A command to assign or remove attributes).

To store edited or newly entered text, the information contained in the work space must be written to a diskette file using the W or N commands (see List of Commands starting on the following page).

ADVANCE TEXT

A $\begin{bmatrix} \$ \\ n \end{bmatrix}$

PURPOSE

The A command advances text in the work space.

PARAMETERS

n may be a decimal number between 1 and 255, inclusive. If n exceeds the number of lines between the present position of the display and the end of the work space, the display is advanced to the end of the work space.

\$ advances the display to the end of the work space.

NOTES

Entering an A command without parameters advances the text one line. Alternatively, the $\begin{bmatrix} \downarrow \end{bmatrix}$ key may be used to advance text one line at a time.

EDITOR

LIST OF COMMANDS

BACK UP TEXT

B [\$]
[n]

PURPOSE

The B command backs up text in the work space.

PARAMETERS

n may be a decimal number between 1 and 255, inclusive. If n exceeds the number of lines between the present position of the display and the beginning of the work space, the display is positioned at the beginning of the work space.

\$ backs up the display to the beginning of the work space.

NOTES

Entering a B command, without parameters backs up the text one line. Alternatively, the key may be used to back up text one line at a time.

CLEAR

CLR

PURPOSE

The CLR command clears the work space and, consequently, the display. The information in the work space is lost.

DELETE

DEL[n]

PURPOSE

The DEL command deletes the Edit Line and n-1 lines following the Edit Line.

PARAMETERS

n may be a decimal number between 1 and 255, inclusive. If n is not specified, only the Edit Line is deleted.

END OUTPUT FILE

E

PURPOSE

The E command terminates use of the output file previously specified in the last W command. The E command is convenient for writing to multiple output files.

EDITOR

LIST OF COMMANDS

FIND AND REPLACE

F[{d}{string}{d}]

F[{d}{string}{d}][{rstring}{d}[A][V]]

PURPOSE

The F command instructs the Editor to search from the Edit Line to the end of the work space for a match between string and a character or series of characters in the work space.

PARAMETERS

d (delimiter) is any character other than a space that does not occur in string or rstring.

string is any series of display characters (characters which are visible on the display screen), including spaces.

rstring (replacement string) is a series of display characters that will replace string. A null replacement string (the second and third delimiters are typed with no intervening spaces) may be used to delete characters from the work space.

A (all) specifies that all matches of string from the Edit Line to the end of the work space be replaced.

V (verify) specifies that the Editor ask for verification before replacing a string.

NOTES

If F is entered without parameters, the previously specified string is used. In this case, the replace option is not activated.

The F command does not differentiate between upper and lower case characters.

FIND AND REPLACE (continued)

EXAMPLES

The following three commands are legal F commands:

- F RETURN - Finds the string specified in a previous F (with string) command.
- F,OLDSTRING TO BE DISPLAYED,
- Finds the next occurrence of OLDSTRING TO BE DISPLAYED
- F/BAD/GOOD/AV - Finds all occurrences of BAD and replaces them with GOOD. The F command asks for verification for each case.

The command:

F.TEST.

finds two matches in the string:

THIS IS A TEST OF THE FITTEST..

Not only does the Editor find the word TEST; it also matches the last syllable of FITTEST. Matches of this kind can be prevented by entering spaces on either side of the string:

F. TEST .

EDITOR

LIST OF COMMANDS

GET AND REPLACE

G[{d}{string}{d}]

G[{d}{string}{d}][{rstring}{d}[A][V]]

PURPOSE

The G command searches the workspace beginning at the Edit Line and the remainder of the previously specified input file for the specified string.

PARAMETERS

d (delimiter) is any character other than a space that does not occur in string or rstring.

string is any series of display characters, including spaces.

rstring (replacement string) is a series of display characters that will replace string. A null replacement string (the second and third delimiters are typed with no intervening spaces) may be used to delete characters from the work space.

A (all) specifies that all matches of string from the Edit Line to the end of the work space be replaced.

V (verify) specifies that the Editor ask for verification before replacing a string.

NOTES

If G is entered without parameters, the previously specified string is used. In this case, the replace option is not activated.

The G command does not differentiate between upper and lower case characters.

If the end of the work space is reached during a Get command, the entire contents of the work space are written (appended) to the current output file (specified in the last W with filename command). The work space is then filled with more text from the current input file (specified in the last L with filename command), and the search continues until a match is found or until the end of the input file is reached.

INSERT LINE

I

PURPOSE

The I command invokes Line Insert Mode, which allows the Editor to accept new text lines. The cursor is moved to the Edit Line.

NOTES

Two successive depressions of `RETURN` instruct the Editor to leave Line Insert Mode and move the cursor to the Command Line.

INSERT CHARACTER

`INS`

PURPOSE

The `INS` key invokes Character Insert Submode (a submode of line Insert Mode) which allows characters to be inserted or deleted within a line without retyping the rest of the line,

NOTES

To enter Character Insert Submode, press the `INS` key once; to exit, press `INS` again. When in this submode, an asterisk is displayed on the dashed line below the Edit Line at the point where new characters will be inserted.

EDITOR

LIST OF COMMANDS

LOAD

L[filename]

PURPOSE

The L command loads the work space with text from the input file (specified in the last L with filename command) until the work space is full or until the end of file is reached.

NOTES.

If the work space contains text, the L command appends the file to the end of that text. If the L command terminates due to a full work space, a message is generated. After a portion or all of the text has been written out, an L (without filename) command may be entered to load the remainder of the file. If an L command is executed when the work space is full, the command is ignored, an error message is generated, and the work space remains unaltered.

NEXT

N

PURPOSE

The N command writes the data contained in the work space to the previously specified output file (W with filename command) and loads the work space from the previously specified input file (L with filename command).

NOTES

This command is useful for editing lengthy files which exceed the capacity of the work space. Successive N commands may be entered until the end of file is reached.

SET LOWER CASE

SL

PURPOSE

The SL command allows entry of both upper and lower case characters.

SET UPPER CASE

SU

PURPOSE

The SU command displays all keyboard entered alphabetic characters in upper case whether or not the shift key is used.

TAB CLEAR AND SET

T[column number][,column number]...

PURPOSE

The T command is used to clear or set tab stops.

PARAMETERS

Tab stops may be set by entering the desired number in the column number parameter(s). column number may be an integer between 2 and 80, inclusive.

NOTES

Entering a T command without parameters clears all tab stops.

Tab stops must be set in numerical order. Tab stops are shown by single vertical lines on the Tab Line of the display.

EDITOR

LIST OF COMMANDS

TEXT BLOCK COMMANDS

Six text block commands are available. Each is defined here in the order of use.

PLACE MARKER

P

PURPOSE

The P command temporarily places a marker in the work space which defines a text block boundary for subsequent copying, moving or deletion.

NOTES

The marker is placed above the Edit Line. Two markers delineate a text block. Markers may be redefined, but no more than two markers may exist at a time; setting a third marker automatically clears the first. The block is displayed in reverse video. Either the top or bottom marker may be set first.

Most commands that modify text within the workspace automatically clear both markers. Commands which do not modify the text, such as cursor movement commands, do not affect the markers.

TEXT BLOCK COMMANDS (continued)

COPY

C

PURPOSE

The C command copies a text block (previously set by the P command) immediately above the current Edit Line.

NOTES

The C command does not delete the original block, nor does it clear the markers.

A text block may be copied repeatedly in any location (except within the text block) by entering a sequence of C commands. If the work space becomes full, the message "OVERFLOW" is displayed. The user must then create free space (via a W\$ or CT command) before the Copy can be executed.

COPY FROM

CF[filename]

PURPOSE

The CF command allows text from a specified file to be merged into the work space immediately before the current Edit Line.

NOTES

The first time a CF command is entered, a filename must be specified.

If the entire file will not fit into the work space, the message "OVERFLOW" will appear on the display screen. The work space contents may then be written out (W\$ or CT command) and the remainder of the file may be loaded by entering CF (without filename). This process may be repeated as necessary.

EDITOR

LIST OF COMMANDS

TEXT BLOCK COMMANDS (continued)

COPY TO

CT[filename]

PURPOSE

The CT command copies a text block (previously set by the P command) to the specified file. If a filename is specified, any previous file contents are overwritten.

NOTES

The first time a CT command is entered, a filename must be specified. Subsequent CT (without filename) commands will append text block data to the file.

The CT command does not delete the text block(s) from the work space, nor does it clear the markers.

The CT command does not affect other write or load operations (W, L or N commands).

It is illegal to enter a CT (without filename) command after a CF (without filename) command or vice versa.

KILL

K

PURPOSE

The K command deletes a text block from the work space and clears the markers.

NOTES

If the current Edit Line lies within the deleted text block, the first line after the block will become the new Edit Line.

TEXT BLOCK COMMANDS (continued)

MOVE

M

PURPOSE

The M command removes a text block from any location in the work space, inserts it above the Edit Line, and clears the markers.

NOTES

The M command does not duplicate text; instead it uses an algorithm which rotates the text in the work space. Thus, it is possible to execute a Move even when the work space is full.

EDITOR

LIST OF COMMANDS

WRITE

W[\$][*][filename]

PURPOSE

The W command writes the work space contents to the output file.

PARAMETERS

If \$ is specified, text from the beginning of the work space to (but not including) the Edit Line is written to the output file.

If * is specified, text written to the output file will also remain in the work space.

If a filename is specified, the W command writes to the beginning of the file and overwrites any previous contents. If a filename is not specified, the W command appends the contents of the work space to the output file specified in the most recent W (with filename) command.

NOTES

The specified filename becomes the new output filename used when subsequent N, G or W (without filename) commands are executed.

TABLE 3-A.
KEYBOARD EDITING OPERATIONS

KEYBOARD COMMAND	COMMAND LINE	LINE INSERT MODE (Indicated by row of = characters under Edit Line)	LINE EDIT MODE (Indicated by cursor on Edit Line)	Character Insert Submode INS (Indicated by * under cursor position on Edit Line)
RETURN	After a command, RETURN enters the command. RETURN without a command moves cursor to Edit Line.	Enters characters typed on Edit Line. Depressing RETURN key twice causes Editor to leave Line Insert Mode and return to Command Mode.	Editor leaves Line Edit Mode and returns to Command Mode.	Editor leaves Character Insert Submode. Continues in Line Insert or Line Edit Mode.
Space Bar	Enters a blank under blinking cursor and moves cursor right one space.	Same	Same	Moves cursor, character under cursor, and all of line to right of cursor right one space. Inserts blank at former cursor position.
BACKSPACE	Moves cursor left one space and removes character under cursor.	Same	Same	Removes character to left of cursor. Cursor and all of line to right of cursor move left one space.
← →	Moves cursor left or right. Does not affect text. Wraps around from column 80 to column 1.	Same	Same	Same.
↑ ↓	Scrolls text through Command Line.	Editor returns to Command Mode.	Scrolls text. Cursor remains on Edit Line.	Editor leaves Character Insert Submode and text is scrolled.
TAB	No function.	Positions cursor at next tab stop.	Same as Line Insert.	Same as Line Insert.
DEL	No function.	Removes character under cursor and moves all of line to right of cursor left one space.	Same as Line Insert.	Same as Line Insert.
CAN	Deletes characters on Command Line and moves cursor to beginning of line.	Same as Line Insert.	Same as Line Insert.	Removes line and moves cursor to beginning of line.

END PREVIOUS FILE?

An attempt was made to write to a new output file without ending use of the current output file (E command). To end the previous file, type Y; to cancel the write command, type N or **RETURN**.

END FILE

The Editor has reached the end of the input file.

FILE NOT SPECIFIED

An L, W, N, G, CT or CF command was entered without specifying a filename, and input or output files were not previously specified. Retype the L or W command and include the desired filename, or execute a W command with a filename before retyping the N or G command.

NOT FOUND

During an F or G command operation, a character string match was not found in the range of search.

NOT A SOURCE FILE

The file designated in an L, W, CT or CF command does not have an S attribute. Use the Manager's A command to change the file's attribute(s).

OVERFLOW

The work space is full. All or part of the work space should be written to the output file using an N or W command.

REPLACE THIS LINE?

The F or G command requires verification before replacing a string in the line displayed. Type Y to replace, N, **RETURN** **TAB** or **STEP** to continue without replacement, or **BREAK** to cancel execution of the command.

SAVE DATA?

An attempt was made to jump to another ADS system program without writing the work space contents to an output file. Type N to erase the text in the work space and allow the program jump. To cancel the jump command and preserve the work space text, press any key other than N.

SYNTAX

There was a syntax error in the last command entered. Retype the command.

4.1 INTRODUCTION

The Assembler translates mnemonics (program instructions in a form easily understood by the programmer) to the machine language instructions necessary for program execution. The Assembler implements a standard set of rules for use, options, and Assembler directives which are applicable to several microprocessors. In most cases, the microprocessor instruction mnemonics and operand formats are identical to those specified by the microprocessor manufacturer.

Assembler Input.

The Assembler accepts assembly language source code from a diskette file and, optionally, macro definitions from a macro library file. The Assembler reads and evaluates the input file one line at a time. Each line input to the Assembler must fall in one of the following categories.

- 1) An assembly language instruction is of the form:

```
[label]{op-code}[operand][,operand]...[;comment]
```

An optional label may be provided by the programmer. The instruction mnemonic is specified by the microprocessor manufacturer. An operand field usually designates either a microprocessor register, a memory address or both. An optional comment usually contains explanatory information. Each instruction causes the Assembler to output one or more bytes of object code. For further information, see Section 4.8.

- 2) An Assembler directive. There are two types of Assembler directives:
 - a) those which cause the Assembler to perform the clerical tasks of assembly;
 - b) macro directives and conditional assembly directives. For further information, refer to Sections 4.9 and 4.10.
- 3) A comment line. The Assembler treats all characters to the right of a semicolon or any line with an asterisk in column 1 as a comment statement.

An example Assembler input sequence is shown in Figure 4-1 on the following page.

ASSEMBLER

<u>Label</u> <u>Field</u>	<u>Op-Code</u>	<u>Operand</u>	<u>Comments</u>
	SPC		
	RSEG	COPYM	;Start RSEG COPYM
	SPC		
COPY	LXI	H,MSGF	;Entry point, load "from" address
	LXI	D,MSGT	;Load D,E with "destination" address
	MVI	C,16	;Load C register with count
	SPC		
LOOP	MOV	A,M	;Load A from MSGF
	STAX	D	;Store A to MSGT
	INX	H	;Increment MSGF pointer
	INX	D	;Increment MSGT pointer
	DCR	C	;Decrement count
	JNZ	LOOP	;If count not 0, jump to LOOP
	SPC		
	RET		;Return to main program
	SPC		
	END		;End of this assembly

Figure 4-1. Example Assembler Input

Assembler Output.

The Assembler generates an output file of relocatable object code which may be processed by the Linker. The Linker combines the file with other relocatable files and assigns absolute memory addresses. A symbol table may be appended to the output file if the user wishes to reference memory addresses via label names during execution.

The Assembler may also print and display a program listing containing each input line, the hexadecimal representation of the object code generated by that line, and other information.

The example input shown in Figure 4-1 produces the listing output shown in Figure 4-2. The first column contains the hexadecimal addresses within the program segment. The second column contains the object output in hexadecimal.

<u>Loc.</u> <u>Counter</u>	<u>Assembler</u> <u>Hex.Code</u>	<u>Label</u> <u>Field</u>	<u>Op-Code</u>	<u>Operand</u>	<u>Comments</u>
					* *Routine to copy a message *
0000			RSEG	COPYM	;Start RSEG COPYM
0000	210000	COPY	LXI	H,MSGF	;Entry point, load "from" addr
0003	110000		LXI	D,MSGT	;Load D,E with "destination" addr
0006	0E10		MVI	C,16	;Load C register with length count
0008	7E	LOOP	MOV	A,M	;Load A from MSGF
0009	12		STAX	D	;Store A to MSGT
000A	23		INX	H	;Increment MSGF pointer
000B	13		INX	D	;Increment MSGT pointer
000C	0D		DCR	C	;Decrement count
000D	C20B00		JNZ	LOOP	;If count not 0, jump to LOOP
0010	C9		RET		;Return to main program
0011			END		;End of this assembly

Figure 4-2. Example Assembler Output

For each assembly, a reference list is output showing the memory address of each global (G) and local symbolic label name and each external reference (X) (see Figure 4-3).

COPY	0000	COPYM	G0000	LOOP	000B	MAINPRG	G0000
MSGF	X0000	MSGT	X0000	ZERO	X0000		

Figure 4-3. Example Assembler Reference List.

For more information, see Section 4.8.

ASSEMBLER

4.2 PROGRAM SEGMENTS

Assembly language programs may be divided into program segments. Each relocatable segment (RSEG) may be named by the user for easy identification. A maximum of eight RSEGs and one absolute segment (the ASEG) are allowed in each assembly. RSEGs may be loaded at any location in memory. ASEG object is loaded at the location(s) specified in the Assembler source program.

4.3 LOCATION COUNTERS

The location counter points to the next available memory location in the current program segment. The counter is incremented by one as each byte of object code is output. Each assembly contains a separate counter for each RSEG and for the ASEG.

4.4 ASSEMBLER OPTIONS

Upon entry to the Assembler, a list of options will appear on the display screen. Options may be selected in any order by typing the appropriate letter(s) as listed in Table 4-A.

TABLE 4-A. ASSEMBLER OPTIONS

Option	Description
L	Display a program listing.
T	Truncate lines of the display to 80 characters and limit printer or display listing of the DC directive to one line. If this option is not specified, all lines generated by the DC directive will be output one byte per line.
E	Display only lines containing errors flagged by the Assembler for display or printing.
S	Append the table of symbolic address labels to the end of the relocatable object file.
Z	Flag all lines containing non-8080 instructions (this option is used with the Z80 Assembler only). Non-8080 lines are flagged with the message: INVALID OP CODE. The Z option is useful for verifying Z80 and 8080 program compatibility.

4.5 SPECIFYING ASSEMBLER FILES

Refer to Figure 4-4. After the desired options have been selected and entered, the Assembler will generate a series of prompts, as listed below. The user may respond to each with any legal file specification as shown in Figure 4-4 (see Sections 1.5, 1.6 and 1.7 for allowable filenames, prefixes and parameters).

- 1) SOURCE FILE - Any filename entered must have the S attribute.
- 2) MACRO FILE - This input is optional (enter **RETURN** to bypass). See Section 4.10 for additional information on MACRO files.
- 3) OUTPUT FILE - This input is optional (enter **RETURN** to bypass). Any filename entered must have the R attribute (a new file created by the Assembler will automatically be assigned the R attribute).
- 4) LISTING FILE - This input is optional (enter **RETURN** to bypass). Any filename entered must have the S attribute or none at all.

```

----- 8080/85 ASSEMBLER V02.0 ----- futuredata -----
SPECIFY ASSEMBLER OPTIONS:
(L) - LIST ON SCREEN
(T) - TRUNCATED LIST
(S) - INCLUDE SYMBOL TABLE
(E) - LIST ERRORS ONLY
) L E
SOURCE FILE:
> I: DEMO. S
MACRO FILE:
)
OUTPUT FILE:
> I: DEMO. R
LISTING FILE:
> P:

```

Figure 4-4. Example Assembler Display.

ASSEMBLER

4.6 ASSEMBLER PROCESSING

After the last filename has been entered and the **RETURN** key has been pressed, assembly begins. Statements are processed in two passes. On the first pass the Assembler assigns values to symbolic labels and stores them in a symbol table. During pass 2 the Assembler analyzes each statement from the input file and generates relocatable object output. If the Assembler recognizes an error, the statement is flagged and zero-bytes are generated for that statement.

4.7 HALT ASSEMBLY

To halt the assembly process at any point, press the **RETURN** key. This is useful for viewing the Assembler output as it scrolls on the display. The Assembler will pause and prompt:

CONTINUE?

To continue, type:

Y

To abort assembly, type:

N or **BREAK**

4.8 ASSEMBLER STATEMENT SYNTAX

Statements are composed of four components or fields. Fields are separated by a space or a series of spaces. Statements are entered one per line with a maximum of 80 characters.

Label Field.

The optional label field may contain a symbolic name which is used to reference the statement or which will be assigned a value by an EQU directive. Labels must begin in the first character position in the line. Labels may be from one to eight characters in length. The first character must be a letter (A-Z) or a dollar sign (\$). Subsequent characters must be letters, dollar signs or numbers (0-9).

During assembly a value is assigned to each label, as described below.

- 1) ASEGs. In ASEGs, the value assigned to a label is the absolute address at which the assembled statement will be stored. This value is identical to the value in the absolute segment location counter, which appears on the left side of the assembly listing.

- 2) RSEGs. In RSEGs, the value assigned to a label is the local address within the segment. This value is identical to the value in the location counter of that segment, which appears on the left side of the assembly listing. When the segment becomes part of an executable program during Linker processing, the Linker adds the absolute address of the first byte in the segment to the local address of the label. The result is the absolute address assigned to the label.
- 3) When the EQU Assembler directive is encountered, the label is assigned the value of the operand field. Further information on Assembler directives is given in Section 4.9.

The Assembler writes each label name, the local address of the label, and other information into a symbol table. Label names are represented in ASCII, with each character requiring one byte of memory.

Operation Code Field.

The operation code (or op-code) field contains an instruction mnemonic (assigned by the microprocessor manufacturer), an Assembler directive or a macro name. This field must begin in the second character position on the line or thereafter and may be no more than six characters in length. Each instruction causes the Assembler to generate zero or more bytes of relocatable object code. An opcode must be present in all assembly input lines, except comment lines.

Operand Field.

The operand field contains additional parameters (if any) associated with the instruction or directive in the opcode field. Operands are separated by commas. The operand field begins after the opcode field, but the two fields must be separated by at least one space. A space terminates the operand field. If a space is inadvertently included in the field, the portion to the right of the space will be read as a comment.

Five types of operands are used in the operand field. An operand expression may contain zero, one, several or all of these types.

- 1) Decimal Constants. Decimal constants are numbers containing digits ranging from 0 to 9. Single-byte decimal constants may range in value from 0 to 255; 2-byte decimal constants may range in value from 0 to 65,535.
- 2) Hexadecimal Constants. Hexadecimal constants consist of the digits 0 to 9 and the letters A to F. The digits must be enclosed in single quotes, and an X must prefix the first quote. Single-byte constants may be one or two hexadecimal digits. Two-byte (memory address) constants may be from one to four hexadecimal digits in

length. When less than the maximum number of digits is used, the constant is treated as if leading zeros were supplied.

Examples of single-byte hexadecimal constants are shown below:

X'1' (same as X'01')
X'A3'

Examples of 2-byte (address) hexadecimal constants are shown below:

X'103' (same as X'0103')
X'BF2A'

- 3) ASCII Character Constants. A character constant consists of one or two characters enclosed in single quotes. Each character is given an 8-bit (1-byte) value. A string of three or more ASCII characters may be used as an operand only with relational operators. If one operand of a relational operator is a string and one is another type of operand, the string is always greater. If both operands of a relational operator are character strings, comparison is done on a byte-by-byte basis using the ASCII collating sequence for determining the comparison results.

Examples:

'AB' is assembled to X'4142'
' ' (space) is assembled to X'20'

- 4) Symbolic Labels. During Assembler pass 1 each statement label is stored in the symbol table along with the location counter value. During pass 2 the value assigned to a label is substituted wherever it occurs as an operand.
- 5) Asterisks. An asterisk used as an operand causes the Assembler to use the location counter value of the first byte of the statement in which the asterisk occurs.

Expressions are evaluated from left to right, resulting in a 16-bit (2-byte) quantity. However, the Assembler directive or microprocessor instruction (in which the expression appears as an operand) may truncate the expression to eight bits or may reverse the order of the bytes.

All expressions must reduce to one of the following forms:

- a) an absolute value;
- b) a relocatable value + an absolute value;
- c) an external value + an absolute value.

There is one exception. An absolute value expression may contain a pair of relocatable terms if the terms are in the same segment and if, when subtracted, the result is greater than zero. Relocatable terms cannot be added.

The individual operands (terms) in an expression are connected by any of the following types of operators: arithmetic, logical, register shift, relational, byte extraction or byte order.

1) Arithmetic Operators.

+	Addition.
-	Subtraction or unary minus sign.
*	Multiplication.
/	Integer division. Remainder is truncated.
n.MOD.m	Remainder when dividing n by m. n and m must be integers; n and m may be expressions.

2) Logical Operators.

.AND.	Performs bit-by-bit logical AND.
.OR.	Performs bit-by-bit logical OR.
.NOT.	Complements each bit of the following 1-byte operand or the low order byte of a 2-byte operand.
.XOR.	Performs bit-by-bit exclusive-OR operation (a or b, but not both).

3) Register Shift Operators.

.SHL.n	Shifts the previous 1-byte operand or the lower byte of a 2-byte operand or expression n bits to the left. The left-most n bits are lost. n may be an expression.
--------	---

ASSEMBLER

`.SHR.n.`

Shifts the previous 1-byte operand or expression n bits to the right. This is an arithmetic right shift. The highest order bit is considered the sign bit and is not shifted or changed. If the highest order bit is a one, ones are shifted into the bit positions vacated by the right shift. If the highest order bit is a zero, zeros are shifted into the vacated bit positions. The rightmost n bits are discarded.

4) Relational Operators.

`n=m`

Equality operator. True if n is equal to m . If the relation is true, the 16-bit result is set to one. If false, the result is zero.

`n>m`

Greater than operator. True if n is greater than m .

`n<m`

Less than operator. True if n is less than m .

`n>=m`

Greater than or equal to. The $>$ and $=$ signs may be in any order.

`n<=m`

Less than or equal to. The $<$ and $=$ signs may be in any order.

`n><m`

Not equal to. True if n does not equal m . The $>$ and $<$ signs may be in any order.

5) Byte Extraction Operators.

H(expression) Extracts the high order byte.

L(expression) Extracts the low order byte.

The high or low order eight bits of an expression may be extracted by enclosing the operand in parentheses and prefixing it with H (high) or L (low). These operators are used when an expression having a 16-bit value must be truncated to an 8-bit value. For example:

H(X'2E35'-2) Evaluates to X'2E'.

L(X'2E35'+2) Evaluates to X'37'.

H(ALPHA) evaluates to the high order eight bits of the value assigned to the symbolic label ALPHA. Byte extraction is not allowed as a term of an expression.

6) Byte Order Operators.

A(expression) High order byte is stored first.

B(expression) Low order byte is stored first.

The byte order of the 16-bit result of an expression evaluation may be specified by enclosing the expression in parentheses and prefixing it with the letter A or B. If a byte order operand is not used, the high order byte is stored first.

A byte order operator in the operand field of a define constant (DC) directive, (see Section 4.9) causes that directive to store both bytes of the expression it encloses. A byte order operator may not be used as a term of an expression.

Operators with the highest precedence are evaluated first. All operators are evaluated from left to right. The precedence is shown below in decreasing order:

*, /, .MOD., .SHL., .SHR.

+, -, unary -

=, >, <, >=, <=, <<

.NOT.

.AND.

.OR., .XOR.

ASSEMBLER

Comment Field.

The comment field allows the programmer to document the action or purpose of a statement. This field is not processed by the Assembler, but is printed or displayed in the program listing. The first character of a comment field must be a semicolon. The semicolon terminates Assembler processing of the line in which it appears. The semicolon and those characters to the right of the semicolon are treated as a comment. The comment field may begin after the operand field if the two fields are separated by at least one space.

Comment Statement.

If an asterisk is in the first character position on a line, the entire line will be treated as a comment. Comment statements are useful for documentation requiring more space than is available in a comment field and for lengthy descriptions such as a program function overview. A line containing a semicolon as the first non-space character is processed as a comment statement. In a macro library file, comment statements which are not inside macro definitions terminate Assembler processing of the file.

4.9 ASSEMBLER DIRECTIVES

Eight types of Assembler directives are available.

Segment Directives.

Either an ASEG or an RSEG directive may appear in an assembly, preceding statements which generate object output. If segment directives do not appear, statements will be assembled into the ASEG.

The ASEG directive defines the beginning of the absolute program segment. Neither a label nor operands are allowed in the directive. The Assembler begins the ASEG at location zero or at a location defined in an immediately following ORG directive. By writing another ASEG directive, it is possible to assemble into the ASEG after intervening segments. An ASEG ends at the beginning of another segment or at the end of an assembly.

The RSEG directive defines a relocatable program segment. A maximum of eight RSEGs are allowed in an assembly, each identified by a user-supplied name in the operand field. A label is not allowed. An RSEG directive is entered as:

```
RSEG{rseg-name}
```

An RSEG ends at the beginning of another segment or at the end of an assembly. rseg-name is automatically made a global symbol. It is possible to assemble into a segment after intervening segments by writing an ASEG directive or an RSEG directive with the appropriate rseg-name.

Segment Origin Directive.

A segment origin directive is entered as:

```
ORG {expression}
```

The origin directive specifies the segment location at which subsequent statements are assembled and sets or resets the location counter of the program segment in which it occurs. The expression must be absolute if in an ASEG, or relocatable if in an RSEG. The ORG directive may appear anywhere in a program segment. If the ORG directive is not used, the segment begins with the location counter set to zero. The location counter is incremented by one as each byte of object code is generated. Symbols appearing in the operand field expression must have been previously defined.

Global Directive.

A global directive is entered as:

```
GLBL {symbol}[,symbol]...
```

A global symbolic reference refers to a common memory location for other assemblies as well as the current assembly. The assembly in which a global symbol is defined and each assembly referencing the global symbol must declare that symbol to be global, using a GLBL directive. In the Assembler symbol table listing, global symbols appearing in the label field of the current assembly are flagged with the letter G. Global symbols used in operand fields are known as external symbols. They are defined outside the current assembly and are flagged with the letter X. All other listed symbols are local.

A maximum of 255 global labels are allowed in one assembly. Relocatable segment names and the entry points specified in the END directive are automatically made global and may not appear in a GLBL directive.

END Directive.

An END directive is entered as:

```
END [expression]
```

The END directive terminates Assembler operation and, optionally, specifies an entry point at which program execution may begin. If a symbolic label name appears in the expression, it automatically becomes global. The END directive is the last statement processed in an assembly input file. If the END directive is missing, an error is flagged.

EQU Directive.

An EQU directive is entered as:

```
{label}EQU{expression}
```

The EQU directive assigns the value of the expression in the operand field to a symbolic name in the label field. The expression is evaluated during Assembler pass 1. The symbol name may be referenced in the operand fields of statements occurring before and after the EQU directive. However, symbols in the operand field expression must have been previously defined. If a symbol in the operand field has not been previously defined, the line will be flagged with the message INVALID FORWARD REFERENCE.

Define Constants Directive.

A Define Constants (DC) directive is entered as:

```
[label]DC{expression}[ ,expression]
```

The DC directive places data in memory beginning at the current value of the location counter. Multiple operands are stored in successive memory locations. The DC directive is the only directive that generates object output.

Data are stored in five ways which may be combined in one DC statement:

- 1) The 16-bit value of an expression is truncated to the low eight bits and stored in one byte of memory.
- 2) The 16-bit value of an expression is truncated to either the low order or high order eight bits by a byte extraction operator and stored in one byte of memory.
- 3) The 16-bit value of an expression prefixed with a byte order operator is stored in two succeeding bytes. Byte order is determined by the operator.
- 4) An ASCII character string is stored one byte per character. The string is enclosed in single quotes. A single quote within the string is represented by two successive single quotes.
- 5) Decimal integers from 0 to 255 (hexadecimal 0 to FF) are stored in one byte. Decimal integers from 256 to 65,535 (hexadecimal 100 to FFFF) are truncated to the lower order eight bits.

The following example demonstrates each type of data storage:

```
LOC  DC  JUMP,L(R+8),B(LOOP),'ABC',12,A(X'A030')
```

Ten data bytes are stored in memory beginning at location LOC. The first byte stored is the low order byte of the 16-bit memory address JUMP. The next byte stored is the low order byte of the 16-bit result obtained when eight is added to memory address R. Then, both bytes of the 16-bit memory address LOOP are stored. Byte order operator B causes the low order byte to be stored first. The ASCII representation of the letters ABC is stored in the next three bytes. The decimal integer 12 is stored in the next byte. Then, the hexadecimal number A030 is stored in two successive bytes. Byte order operator A causes the high order byte to be stored first.

Define Memory Space Directive.

A Define Memory Space (DS) directive is entered as:

```
[label]DS{expression}
```

The DS directive reserves memory space by advancing the segment location counter. The amount of space desired (in bytes) is entered in the operand expression. If a symbol name is used as the expression or a term of the expression, it must have been previously defined; if not defined, the line will be flagged with the message: INVALID FORWARD REFERENCE.

Printer Control Directives.

- 1) SPC. This directive places a blank line in the listing for improved readability. No operands are processed.
- 2) EJE. The eject directive causes the Assembler to skip to the top of the next page of the listing for improved readability. No operands are processed.
- 3) PRINT OFF. PRINT OFF suppresses printing of all following source lines. The directive itself is not printed.
- 4) PRINT ON. PRINT ON lists all source lines except:
 - a) lines generated in a macro expansion;
 - b) lines skipped due to conditional assembly directives;
 - c) conditional directives themselves.

Lines are printed after set-symbol substitution has taken place.

ASSEMBLER

- 5) PRINT GEN. PRINT GEN lists macro expansion lines as well as the lines listed by PRINT ON. Macro expansion lines are printed after parameter substitution and set symbol substitution have taken place. This directive affects printing and the display listing identically.
- 6) PRINT ALL. PRINT ALL prints all source lines including lines skipped due to conditional assembly directives and the conditional assembly directives themselves. Location counter values and object data for skipped lines are not printed. Lines are printed after set symbol and macro parameter substitutions have taken place. This directive affects printing and the display listing identically.

4.10 MACROS

The macro facility enhances Assembler capability by allowing a single calling statement to generate groups of instructions. As shown in Figure 4-5, a macro definition begins with a "MACRO" pseudo-op and ends with an "ENDM" pseudo-op.

Since it is more common to generate a group of similar rather than identical instructions, the macro statements may contain parameter names which are referenced and replaced by actual values. These parameter names are listed (separated by commas) in the operand field of the MACRO pseudo-op. Each macro parameter name may be referenced within the body of a macro definition by a statement containing the macro parameter name preceded by an ampersand. When a calling statement is encountered, the instructions contained in the macro definition are assembled and the parameter values specified in the calling statement are substituted for each ampersand and name.

Macro name	MACRO	parameter1,parameter2	MACRO pseudo-op)
	statement		
	statement & parameter1		(macro body)
	.		
	.		
	.		
	ENDM		(ENDM pseudo-op)
	.		
	.		
	.		
	Macro-name		(calling statement)

Figure 4-5. MACRO Definition.

An example of macro operation is shown below.

	TEST	MACRO	ABC	(MACRO pseudo-op)
Macro		LXI	H,&ABC	(macro parameter reference)
Definition		ENDM		(ENDM pseudo-op)
Macro	+	TEST	'AB'	(Calling statement)
Expansion		LXI	H,'AB'	(Statement generated when macro is assembled)

Syntax.

A macro name must begin in the first character position of the MACRO pseudo-op. Macro names and macro parameter names (in MACRO pseudo-ops) must begin with a letter (A-Z) or a dollar sign (\$). Subsequent characters must be letters (A-Z), numbers (0-9) or dollar signs. Macro names and macro parameter names may be from one to eight characters in length. Commas and blanks may exist in a parameter if placed between single quotes (there must be an even number of quotes in any parameter). Examples of valid parameters are listed below:

```
20
X'40'
'A,B'+ 'B,C'
```

Macro parameters may be from 0 to 32 characters in length, and may contain any character.

ASSEMBLER

Substitution.

Substitution may occur in the label field, opcode field, operand field, comment field or any combination of these fields. For the example macro definition:

```
SAVEA    MACRO    TYPE,LOC
          &TYPE    &LOC
          ENDM
```

the call:

```
SAVEA    STA,VAL
```

results in:

```
STA      VAL
```

The call:

```
SAVEA    STAX,D
```

results in:

```
STAX     D
```

Literal Ampersand.

As previously stated, the ampersand signals the macro processor to substitute a parameter value. At times, however, a literal ampersand character is needed. Four consecutive ampersands are used to tell the macro processor that rather than substituting, an ampersand character is desired. In the example below a single ampersand is stored in a constant.

```
DC    'ABC&&&&XYZ'
```

Placement of Macro Definitions.

Macros must be defined at the beginning of the input source file or macro library file. The Assembler reads macros until it reaches a statement outside a macro definition other than PRINT, SPACE or EJE printer control directives. By specifying a macro file to the Assembler, a second source file of macro definitions (a macro library) may be included in the assembly. Macros may be defined in either the input source file or a macro library file or both.

Duplicate Macro Definitions.

If a macro library is specified, the Assembler reads macro definitions from the library before reading the main source input file. Macros in the main source file take precedence over macros in the macro library. If two macros with the same name are defined, the Assembler uses the last one read.

Generating Unique Labels.

A macro call may generate statements containing labels. For example, the following definition is used to set the DE registers to the absolute value of the DE registers' contents (8080 microprocessor):

```

ABSD      MACRO
          MOV      A,D
          ORA      A
          JP       ENDABSD
          CMA
          MOV      D,A
          MOV      E,A
          CMA
          MOV      E,A
          INX      D
          EQU      *
          ENDM
ENDABSD

```

The first time ABSD is called, label ENDABSD is defined. Since ENDABSD has been previously defined, a duplicate label definition error occurs the second time ABSD is called. To avoid duplication, a unique label must be generated each time the macro is called. The Assembler provides a special predefined macro parameter (INDX) which is set to a unique 5-digit numeric value each time a macro is called. The macro ABSD may now be defined using INDX as shown below:

```

ABSD      MACRO
          MOV      A,D
          ORA      A
          JP       AB&INDX
          CMA
          MOV      D,A
          MOV      A,E
          CMA
          MOV      E,A
          INX      D
          EQU      *
          ENDM
AB&INDX

```

ASSEMBLER

Assuming that this is the only macro defined and that there are exactly two calls made, the first call defines the label AB00001, and the second call defines the label AB00002.

Concatenating Parameters.

In the previous macro example, the value of parameter INDX was appended to the right of the characters AB by writing the parameter name (preceded by an ampersand) to the immediate right of the characters AB: AB&INDX. If the user wishes also to append the letter A to the right of the INDX parameter value, a problem will arise:

```
JP  AB&INDXA
```

The macro processor interprets this to mean: substitute the value for parameter INDXA. To solve this problem, the macro processor recognizes the exclamation mark (!) as the parameter name delimiter indicating the end of a parameter name. The letter A may be appended to the right of parameter INDX as shown below:

```
JP  AB&INDX!A
```

On the first call this statement becomes:

```
JP  AB00001A
```

Note that if ! appears anywhere other than after a macro parameter (or a set symbol), it will be processed as a normal character.

Macro Calls Within Macros.

A macro definition may include a statement which calls another (or the same) macro. These calls may be nested to a level of 127 (depending on the Assembler symbol table space available). A macro definition may not have another macro defined in its body.

EXITM Statement.

When encountered during a macro expansion, the EXITM statement immediately terminates assembly of the current or named macro. The next statement processed will be the first one following the macro call. The correct syntax is shown below. Note that a label is not allowed.

```
EXITM{macro-name}
```

4.11 CONDITIONAL ASSEMBLY

Conditional assembly statements allow the programmer to selectively assemble statements in a source file. The variables used in conditional statements are known as set symbols. A set symbol value may be a number from 0 to 65,535 or a character string containing from 0 to 32 characters. Values are set by use of the DEFL (define local) and DEFG (define global) statements.

DEFL Statement.

This statement is used to define (and redefine) local set symbols. Local set symbols are known only in the macro in which they are defined. A set symbol of the same name defined in another macro is considered to be a different set symbol. The proper syntax for a DEFL statement is shown below:

```
{set symbol name} DEFL {expression}
```

or

```
{set symbol name} DEFL {'string'}
```

If apostrophes are placed around the operand, it is processed as a character string. If apostrophes are not placed around the operand, it is processed as an integer expression. Single apostrophes within strings must be doubled.

Labels may appear in the expression only if they have been previously defined. When defining a set symbol, an ampersand must not precede the name in the label field. However, each set symbol reference must be preceded by an ampersand.

DEFG Statement.

This statement is used to define (and redefine) global set symbols. A global set symbol is known in the entire assembly, including all macros, unless a local set symbol of the same name is defined in a particular macro. In this case, after the local set symbol is defined, the global set symbol is unknown in that macro call. The appropriate syntax for a DEFG statement is shown below:

```
{set symbol name} DEFG {expression}
```

or

```
{set symbol name} DEFG {'string'}
```

ASSEMBLER

If apostrophes are placed around the operand, it is processed as a character string. If quotes are not placed around the operand, it is processed as an integer expression. Quotes embedded in the string must be doubled. A label may appear in the expression only if it has been previously defined.

When defining a set symbol, an ampersand must not precede the name in the label field. However, each set symbol reference must be preceded by an ampersand.

SUBSTR Statement.

The SUBSTR (substring) statement assigns part of a string to a set symbol. If the set symbol has not been previously defined, a new local set symbol will be defined. The proper syntax is shown below:

```
{set symbol name} SUBSTR {expression A}, {expression B}, {'string'}
```

expression A defines the beginning character position of the substring; the first character is position 1. expression B defines the length of the substring. If expression B is zero, the substring will begin with the character defined by expression A and continue to the end of the string.

LENGTH Statement.

The LENGTH statement assigns the length of a string to a set symbol. If the set symbol has not been previously defined, a new local set symbol will be defined. The proper syntax is shown below:

```
{set symbol name} LENGTH{'string'}
```

IF Block.

During Assembler operation, an IF block selects specific sections of code for processing. IF blocks may be nested within IF blocks or DO blocks (see page 4-25) to any level. An IF block must begin with an IF statement and must end with an ENDIF statement. These two statements and the optional ELSEIF, ELSE and EXITIF statements are described below.

- 1) IF Statement. The IF statement begins an IF block. An optional IF statement name may be entered in the label field for referencing by the EXITIF statement. If the expression contained in the IF statement operand field is non-zero, the statements following the IF statement and preceding the first ELSE or EXITIF statement (at the same nesting level) will be processed. If the expression is zero, the statements following the IF and preceding the first ELSEIF, ELSE or ENDIF statement (at the same nesting level) will be ignored. The proper syntax is shown below:

```
[if block name] IF{expression}
```

- 2) ELSEIF Statement. The ELSEIF statement is used in conjunction with an IF statement to test an alternate condition without going to a deeper nesting level. If the expression in the IF statement is zero, the expressions in all previous ELSEIF statements at this nesting level are zero, and the expression in this ELSEIF statement is non-zero, statements preceding the next ELSEIF, ELSE or ENDIF statements (at this nesting level) will be processed. The proper syntax is shown below. Note that a label is not allowed.

```
ELSEIF {expression}
```

- 3) ELSE Statement. The ELSE statement is used in conjunction with an IF statement to indicate the last alternative. The statement is identical to:

```
ELSEIF 1
```

That is, if the expressions in the IF statement and all subsequent ELSEIF statements at this nesting level are zero, the statements after the ELSE statement and preceding the closing ENDIF statement are processed. The correct syntax is shown below. Note that a label and operands are not allowed.

```
ELSE
```

- 4) EXITIF Statement. The EXITIF statement causes all statements preceding the closing ENDIF statement in the named or current IF block to be ignored. The correct syntax is shown below. Note that a label is not allowed.

```
EXITIF [if block name]
```

- 5) ENDIF Statement. The ENDIF statement terminates an IF block. The correct syntax is shown below. Note that a label and operands are not allowed.

```
ENDIF
```

Example. The ADDX macro defined earlier can be expanded to add a number between 0 and 255 to the BC, DE or HL registers as shown on the following page.

ASSEMBLER

```
ADDXY    MACRO    REG, NUM
          IF      '&REG'='BC'
REGH     DEFL    'B'
REGL     DEFL    'C'
          ELSEIF  '&REG'='DE'
REGH     DEFL    'D'
REGL     DEFL    'E'
          ELSE
REGH     DEFL    'H'
REGL     DEFL    'L'
          ENDF
MOV      A, &REGL
ADI      &NUM
MOV      &REGL, A
MOV      A, &REGH
ACI      0
MOV      &REGH, A
ENDM
```

DO Block.

During Assembler operation, the statements contained in a DO block are repeatedly assembled. DO blocks may be nested within IF blocks or DO blocks to any level. A DO block must begin with a DO statement and end with an ENDDO statement. These two statements and the optional EXITDO and NEXTDO statements are described below.

- 1) DO Statement. The DO statement begins a DO block. An optional name may be contained in the label field for referencing by the EXITDO and NEXTDO statements. The expression in the operand field is evaluated once at the entry to the DO block and is stored as the DO count (the number of times the statements within the block are processed). The proper syntax is shown below:

```
[do block name] DO [expression]
```

If expression is omitted, the block will be processed 65,536 times (essentially indefinitely).

- 2) EXITDO Statement. The EXITDO statement causes the Assembler to immediately terminate processing statements in the current or named DO block and begin at the first statement after the closing ENDDO statement. The correct syntax is shown below. Note that a label is not allowed.

```
EXITDO [do block name]
```

- 3) NEXTDO Statement. The NEXTDO statement causes the Assembler to immediately begin processing the next iteration of the current or named DO block. If this is not the last iteration, the next statement processed will be the first statement after the DO statement. The correct syntax is shown below. Note that a label is not allowed.

```
NEXTDO [do block name]
```

- 4) ENDDO Statement. The ENDDO statement terminates a DO block. The correct syntax is as follows. Note that a label and operands are not allowed.

```
ENDDO
```

Example. One use of conditional statements is that of generating tables that would otherwise be tediously entered from the keyboard. The example below generates a table to check for a numeric character.

```
TABLE EQU *
CHAR DEFG 0
      DO 256
      IF &CHAR>='0'.OR.&CHAR<='9'
      DC 0
      ELSE
      DC X'FF'
CHAR DEFG &CHAR+1
ENDDO
```

4.12 ADVANCED FEATURES

During assembly, each source line is scanned twice for ampersands which signal set symbol or macro parameter substitution. During each scan the values of the set symbol names (preceded by ampersands) are substituted for the ampersand and name. Double ampersands are replaced by a single ampersand. Two special types of set symbol specifications are allowed.

Subscripted Set Symbols.

The following statement will define a set symbol whose name is based on the value of another set symbol:

```
A&I DEFL 'A'
```

If the value of &I is the integer 4, the set symbol A00004 will be defined to have the value A.

ASSEMBLER

The following statement uses the generated set symbol:

```
MVI B, '&&A&I'
```

On the first scan, && is replaced by &, and &I is replaced by 00004, resulting in:

```
MVI B, '&A00004'
```

On the second scan, &A00004 is replaced by A, resulting in:

```
MVI B, 'A'
```

By varying the value of set symbol I, set symbols may be defined, referenced and indexed (using I as a subscript).

Indirect Set Symbols.

The following statements define a set symbol whose value is the name of another set symbol:

```
A DEFG 'ABC'  
B DEFG 'A'
```

The following statement uses set symbol B as an indirect reference to the value of set symbol A:

```
IF '&&&B'='ABC'
```

On the first scan, the first two ampersands are replaced by a single ampersand, and &B is replaced by A, resulting in:

```
IF '&A'='ABC'
```

On the second scan, &A is replaced by ABC, resulting in:

```
IF 'ABC'='ABC'
```

LIST OF ASSEMBLER
MESSAGES

DUPLICATE SYMBOL

Respecify.

ERROR WHILE WRITING OBJECT FILE

Terminates Assembler processing.

EXPRESSION HAS MORE THAN 1 RELOCATABLE FACTOR

Expressions with relocatable or absolute terms.

IMMEDIATE VALUE > -128 or < 127

The instruction allows only a signed 8-bit value. The high-order bit is used as the sign. Respecify.

INVALID FORWARD REFERENCE

Example: A EQU B
where B is not yet defined.

INVALID LABEL

Respecify.

INVALID OPCODE

Respecify.

INVALID OPERAND

Respecify.

MISSING END

An END statement must be the last statement in an assembly.

MORE UNPRINTABLE ERRORS

Occurs if there are more than 10 errors in one line.

OVER 255 EXTERNALS

Terminates Assembler processing.

RELATIVE JUMP OUT OF RANGE

Appropriate processors only.

LIST OF ASSEMBLER

MESSAGES

RELATIVE JUMP TO DIFFERENT RSEG

Appropriate processors only.

RELATIVE JUMP TO EXTERNAL SYMBOL

Appropriate processors only.

SYMBOL TABLE OVERFLOW

Terminates Assembler processing.

UNDEFINED SYMBOL.

Respecify.

LIST OF MACRO
AND CONDITIONAL
ASSEMBLY MESSAGES

ELSE OUT OF PLACE
ENDDO OUT OF PLACE
ENDIF OUT OF PLACE
ENDM OUT OF PLACE
EXITM OUT OF PLACE
INVALID MACRO PARAMETER NAME
MACRO DEFINITION OUT OF PLACE
MACRO NESTING EXCEEDS 127
MISSING ENDDO
MISSING ENDIF
MISSING ENDM
OPERAND LONGER THAN 32 CHARS
UNDEFINED SET SYMBOL

5.1 INTRODUCTION

The Linker combines relocatable program segments (RSEGs) from Assembler- or Compiler-generated relocatable file(s) (R attribute) to form a single executable absolute object file. Address references between RSEGs that were unresolved at assembly or compile time are resolved, and RSEGs are relocated for loading at absolute addresses. In addition to RSEGs, the Linker also supports the ASEG, the special program segment for which no relocation is necessary.

5.2 LINKER INPUT

Assembler- or Compiler-generated relocatable object files may be used as input to the Linker. Each relocatable object file may contain a maximum of eight RSEGs and one ASEG. If two or more RSEGs having the same name appear in different input files, the Linker will use the first one encountered. If two or more input files contain the ASEG directive, memory allocations may overlap. If RSEG directives are not used in an assembly, object code will be placed in the ASEG.

In order to link symbolic name references in object files, the name must be declared "global" (using the GBL directive) during the assembly or compilation of each file. For further information on global symbols, refer to Sections 4.9 and 4.10.

5.3 LINKER OUTPUT

The executable absolute object file generated by the Linker may be loaded into memory and executed by the Debugger. In addition, a memory map and reference list may be displayed or printed.

5.4 OPTIONS

Three Linker options are available to choose the desired type of input/output operation. Upon initial entry to the Linker, these options are displayed on the display screen. To select the desired options, type the appropriate letters as listed in Table 5-A. Options may be selected in any order either from the keyboard or from a command file.

LINKER

TABLE 5-A. LINKER OPTIONS

OPTION	FUNCTION
D	Links only those RSEGs which are specifically named in the Linker commands and deletes all other RSEGs. If D is not selected, all RSEGs from all input files will be linked, and the name and length of each RSEG in each input file will be displayed. Segments named in input commands are positioned in memory as requested. All unnamed segments are positioned after the last named segment.
S	Writes symbol tables from all input files to the output file. These tables, which relate global symbols to memory locations, are necessary for symbolic debugging. A symbol table may be placed in each relocatable object file by the Assembler or Compiler.
L	Displays the memory map and reference list.

5.5 COMMANDS

The Linker commands are entered to specify the following information:

- 1) the input file(s);
- 2) the output file;
- 3) the listing file (if any);
- 4) RSEG placement in memory;
- 5) lists of RSEG names in the order desired;
- 6) program execution entry point.

RSEG memory placement (#ORG) commands may cause an RSEG to overlap a previously specified RSEG or the ASEG. An overlap condition is flagged by an "0" in the length column of the memory map.

An example Linker display is shown in Figure 5-1. Option D was not selected. The dashed line marks the end of the display. The next **RETURN** will cause Linker processing to begin.

```
LINKER V02.0                                futuradata
SPECIFY LINKER OPTIONS:
(D) - DELETE UNSPECIFIED RSEGS
(S) - INCLUDE SYMBOL TABLE
(L) - LIST TABLES ON CRT
)LS
SPECIFY INPUT FILE
>I: DEMO.R
RSEG      LENGTH
CODE      0010
DATA      0BBB
SPECIFY INPUT FILE
)
SPECIFY OUTPUT FILE
>O: DEMO
SPECIFY LISTING FILE
>P:
LINKER INPUT
>ORG 'X'1000'
```

Figure 5-1. Example Linker Display

LINKER

Each Linker command is defined below in the order of use.

COMMAND	EXPLANATION
{input file-spec} [input file-spec] . . . <u>RETURN</u>	After the Linker options have been entered, the Linker will prompt: INPUT FILE. Type the desired filename(s), one filename and <u>RETURN</u> per line. Press the <u>RETURN</u> key after the last input file-spec has been entered.
[output file-spec]	The Linker will prompt: OUTPUT FILE. Type the output file specification and press <u>RETURN</u> .
[listing file-spec]	The Linker will prompt: LISTING FILE. Type the listing file specification and press <u>RETURN</u> .
#ORG {absolute address}	The Linker will prompt: LINKER INPUT. Specify the absolute address of the first relocatable program segment in the list following the #ORG statement.
{rseg-a}{[,rseg-b] [rseg-c][,rseg-d]...	Enter a list of RSEG names. The RSEGS are positioned in memory in the order named, starting at the address in the preceding #ORG statement.
#ORG {absolute address} [rseg-e]	Additional starting addresses, each with a list of RSEGS, may be specified.
#END [entry point name]	Enter a #END statement to begin Linker processing. entry point name is a global symbol specifying the program entry point. (RSEG names are automatically global.) If not specified, the program entry point is set to the address of the first RSEG. After processing begins, press the <u>RETURN</u> key to halt Linker processing and view display output. Press <u>RETURN</u> again to continue. When processing is complete, the Linker displays the message: FUNCTION COMPLETED. Press the <u>RETURN</u> key. To begin Linker operation again enter an input file-spec. To jump to another ADS system program, enter a J command.

5.6 MEMORY MAP OUTPUT

An example memory map display is shown in Figure 5-2. The memory map lists:

- 1) RSEG memory locations in the order input to the Linker (if the #ORG commands were input with their operands in ascending order, the listed memory locations will also be in ascending order);
- 2) RSEG names;
- 3) names of the file(s) from which the RSEGs were taken;
- 4) lengths of the RSEGs in hexadecimal.

ADDR	RSEG	FILE	LENGTH
1000	PROGRAM	SKELETON.R	059F
159F	IOCODE	EXEC00.R	002B
15C7	DEVICE	DEVICE.R	0005
15CC	INCODE	INPUT00.R	03BD
1989	KBCODE	KB00.R	0077
1A00	TVCODE	TV00.R	032D
1D2D	NIO	NET00.R	05E5
2292	STACK	SKELETON.R	0064
22F6	DIO	DISK00.R	09CA
2CC0	LPCODE	LP00.R	0360
D600	WADATA	WORK.R	0200
D800	CRT	TV00.R	07D0

LINKER V02.0 PAGE 2
FUTUREDATA

FILE	RSEG	ADDR	LENGTH
SKELETON.R	STACK	2292	0064
	PR		

Figure 5-2. Example Memory Map Output

If an RSEG was deleted (option D), its name will appear in the memory map and the address will be flagged with a "D". If an RSEG is overwritten by another RSEG, the overwritten RSEG will be flagged with an "O" in the memory map.

5.7 REFERENCE LIST OUTPUT

An example Linker reference list display is shown in Figure 5-3. The display lists:

- 1) each disk file from which the RSEGs were input;
- 2) the name of each RSEG;
- 3) the beginning hexadecimal address of each RSEG;
- 4) the hexadecimal length of each RSEG;
- 5) the global symbols with the absolute memory address assigned to each symbol;
- 6) the entry point address at which program execution will begin.

PAGE 2
FUTUREDATA

LINKER V02.0

FILE	RSEG	ADDR	LENGTH
SKELETON.R	STACK	2292	0064
	PROGRAM	1000	059F
GLOBALS			
PROGRAM	1000	STACK	2292
FILE	RSEG	ADDR	LENGTH
EXEC00.R	IOCODE	159F	0028
GLOBALS			
IOCODE	159F	IOEXEC	159F
FILE	RSEG	ADDR	LENGTH
DEVICE.R	DEVICE	15C7	0005

Figure 5-3. Example Linker Reference List Display

****DELETED RSEG REFERENCE IN (filename)**

An RSEG which is present in an input file, but is not included in the output file, is needed to resolve address references.

(label) **DUPLICATE GLOBAL IN (filename)

The first occurrence of a global symbol is used for address references. All additional definitions are flagged as errors.

NOT A RELOCATABLE FILE

The Linker input file was not created by the Assembler or Compiler or the file's "R" attribute was changed with the Manager. Assign the "R" attribute to the file.

PARAM ERR ... RESPECIFY

There is a syntax error in the Linker input or the specified RSEG name was not found.

****RELOCATION ERROR IN (filename)**

The input file was not correctly assembled. Reassemble or recompile.

****SEQUENCE ERROR IN (filename)**

The records in the named file are not in proper order. Reassemble or recompile.

****SYMBOL TABLE NOT FOUND IN (filename)**

The symbol table was not included when the program was assembled or compiled. Reassemble or recompile.

****TABLE OVERFLOW**

The Linker needs more memory space.

(label) **UNRESOLVED REFERENCE IN (filename)

The external reference was not found in the global symbol table.

6.1 INTRODUCTION

The Command File Processor reads command lines from a diskette file and passes them to ADS programs, thus providing a convenient alternative to the time-consuming method of keyboard command entry. The Command File Processor is particularly helpful when long sequences of commands must be entered repeatedly.

6.2 COMMAND FILE CREATION

A command file consists of a number of legal ADS commands (one command per line). Command files may be created with the Editor program. The Editor automatically assigns the S attribute to newly created files (the Editor will work only with S-attributed files).

6.3 INITIATE COMMAND FILE PROCESSING

To transfer control from the keyboard to the Command File Processor, type the following sequence:

```
J^{filename}{[,parameter-1][,parameter-2]...[parameter-9]}
```

filename is the name of the source (S attribute) file in which commands have been stored; the Command File Processor reads commands from this file. parameter-1 is a string of characters which directly replaces the ^1 sequence in the command file; parameter-2 replaces ^2, etc. (a maximum of 9 parameters may be entered). The number of parameters entered at the initiation of command file processing must be equal to or greater than the highest numbered substitution sequence ^n in the command file (see Section 6.4).

Two successive commas (,,) in the initiation command denote a null parameter and cause every occurrence of the corresponding substitution sequence ^n to be deleted and the part of the line to the right of ^n to be moved left two character positions. If the last parameter in an initiation command is null, an extra comma must be entered at the end of the command.

In the following example, four parameters have been entered. The first and last parameters are null. The substitution sequence in the command file (FILEA) must be no higher than ^4.

```
J^FILEA,,FILEX,P,,
```

In some ADS system programs an S option invokes symbol table generation. For instance, if the command file contains:

```
JA          (start Assembler operation)
TL^3       (Assembler options)
```

COMMAND FILE PROCESSOR

the third parameter of the initiation command J^FILEA,,FILEX,S,, causes the options TOS to be entered. If the third parameter is null, the options TO are entered. Thus, the user may choose to assemble with symbol table output at the initiation of command file processing rather than at the time the command file is written.

NOTE

If the Assembler or Linker is used in a command file, a blank line is required at the end of the input sequence to continue command file processing.

6.4 COMMAND FILE PROCESSOR FEATURES

Four features are provided by the Command File Processor, as described below.

^L Feature.

The ^L keyboard escape feature is entered as:

[...part of command line]^L[part of command line...]

This feature allows characters to be entered from the keyboard until the RETURN key is pressed.

Parts of the command file line are combined with characters entered at the keyboard to form a complete command line. The first part of the command line is read from the command file. Input is then taken from the keyboard. When the RETURN key is pressed, the Processor reads the next part of the command from the command file. The keyboard-entered characters are inserted in the command at the ^L position. The RETURN, which ends keyboard input, is not inserted.

If ^L appears by itself in a command file line, the Processor reads only keyboard-entered characters for that line.

In the example below, the Editor's Find command is constructed from parts of a command file line. The characters ^L are replaced with the keyboard-entered characters S20 during execution.

line in command file:

F,R2,^L,AV

keyboard entry for ^L:

S20 RETURN

effective command input to the Editor:

F,R2,S20,AV RETURN

In the next example, the ^L feature allows the user to specify file prefix characteristics (drive numbers and N:, which indicates that a new file must be created).

line in command file:

```
M^L:FILEA,^L:FILEB
```

keyboard entry at first ^L:

```
1 [RETURN]
```

keyboard entry at second ^L:

```
N:0 [RETURN]
```

effective command input to ADS:

```
M1:FILEA,N:0:FILEB [RETURN]
```

^K Feature.

The ^K keyboard escape feature is entered as:

```
[... part of command line]^K
```

This feature allows part of a command and whole command lines to be entered at the keyboard until a caret (^) is entered. The Command File Processor then continues reading from the command file.

The optional part of the command line contains character(s) read from the command file. When these characters are combined with character(s) entered at the keyboard, a complete command line is formed. The user may enter as many commands (terminated by [RETURN]) from the keyboard as necessary.

COMMAND FILE PROCESSOR

An example entry sequence using the ^K feature is shown below.

Line in Command File	Entry at Keyboard	Explanation	Effective Command Input to ADS
JE		Begin Editor Operation.	JE <input type="text" value="RETURN"/>
LTEXT.^K		Display beginning of Load command.	
	6 <input type="text" value="RETURN"/>	Take further input from keyboard. User enters the rest of the filename to be loaded.	LTEXT.6 <input type="text" value="RETURN"/>
	A30 <input type="text" value="RETURN"/>	User edits text.	A30 <input type="text" value="RETURN"/>
	<input type="text" value="RETURN"/>	Other editing commands.	
	.		
	.		
	.		
	DEL2 <input type="text" value="RETURN"/>	User finishes editing text and returns to the command file for further command input.	DEL2 <input type="text" value="RETURN"/>
WTEXTA		The next line from the command file is used.	WTEXTA <input type="text" value="RETURN"/>

^n Feature.

n may be integer from 1 to 9, inclusive. This sequence of characters in a command file is replaced by the nth parameter entered at the initiation of command file processing.

A parameter entered at the initiation of command file processing replaces this sequence by direct substitution (i.e., ^1 is replaced by the first parameter, ^2 by the second, etc.). The line in which the sequence occurs is expanded or contracted to fit the parameter.

For example, the following command file resides on drive 1 and is named TEST.CF.

```
JE
L^1.X
W^2.X
```

This command file can be initiated by the command:

```
J^1:TEST.CF,OLD,NEW|RETURN|
```

which is equivalent to entering

```
JE
L OLD.X
W NEW.X
```

at the keyboard.

^^ Feature.

The double caret passes one caret to the currently active ADS system program. Normally, a single caret invokes a Command File Processor function. A double caret allows a single caret to be passed from a command file to an ADS system program.

To jump to a command file from within a command file, the following command line is used:

```
J^^{filename}{[,parameter1-][,parameter-2]...[parameter-9]}
```

NOTE

filename may be the name of the current command file.

COMMAND FILE PROCESSOR

6.5 KEYBOARD-ONLY INPUT

Special function keys (`[CAN]`, `[DEL]`, `[RESET]`, `[STEP]`, etc.) which do not result in the display of a character must be entered from the keyboard. For example, the `[STEP]` key, which is used during Debugger operation to execute a single microprocessor instruction, cannot be entered from a command file.

6.6 ERRORS

There are two classes of errors which affect command file processing:

- 1) syntax errors;
- 2) errors recognized by ADS system programs which cause command file processing to stop.

Termination of command file processing can be recognized by the following conditions: the display does not change, and the blinking cursor returns to the display.

6.7 ABORT PROCESSING

Command file processing may be aborted by depressing the `[SHIFT]` and `[BREAK]` keys simultaneously.

COMMAND FILE PROCESSOR

LIST OF MESSAGES

FIRST CHARACTER OF FILE NOT "J"

The first character of a command file must be a J command.

PROLOG PROM
PROGRAMMER INTERFACES

A.1 INTRODUCTION

Interfaces for two Prolog PROM Programmer models are available for use with the 2300-ADS Debugger: the 920 and the 900B. Commands and instructions for hardware connections are identical for both models, as described herein.

The files contained on the 920 8080/8085 interface diskette are shown in Table A-1.

TABLE A-1. 920 8080/8085 INTERFACE DISKETTE

Filename	Function
PR92080.N	Prolog interface source file
PR92080.O	Prolog interface source file
PR92080.C	Command file used to edit, exchange, assemble and link PR92080.N
PR92080.R	Prolog interface relocatable file
EIA80.N	Source file of EIA port I/O routines
EIA80.R	Relocatable file of EIA port I/O routines
KI080.R	Relocatable file of CRT routines
KEYIN80.R	Relocatable file of keyboard routines
MACLIB80.N	Macro library (used to generate logo message)
PR92080	Executable object file containing PR92080.R, EIA80.R, KEYIN80.R and KI080.R

PROLOG PROM
PROGRAMMER INTERFACES

Table A-2 lists the files contained on the 920 Z80 interface diskette.

TABLE A-2. 920 Z80 INTERFACE DISKETTE

Filename	Function
PR920Z8.N	Prolog interface source file
PR920Z8.O	Prolog interface source file
PR920Z8.C	Command file used to edit, exchange, assemble and link PR920Z8.N
PR92080.R	Prolog interface relocatable file
EIAZ80.N	Source file of EIA port I/O routines
EIA80.R	Relocatable file of EIA port I/O routines
KI080.R	Relocatable file of CRT routines
KEYIN80.R	Relocatable file of keyboard routines
MACLIB80.N	Macro library (used to generate logo message)
PR92080	Executable object file containing PR92080.R, EIA80.R, KEYIN80.R and KI080.R

The files contained on the 900B 8080/85 interface diskette are shown in Table A-3.

TABLE A-3. 900B 8080/8085 INTERFACE DISKETTE

Filename	Function
PR900B80.N	Prolog interface source file
PR900B80.O	Prolog interface source file
PR900B80.C	Command file used to edit, exchange, assemble and link PR900B80.N
PR900B80.R	Prolog interface relocatable file
EIA80.N	Source file of EIA port I/O routines
EIA80.R	Relocatable file of CRT routines
KEYIN80.R	Relocatable file of keyboard routines
MACLIB80.N	Macro library (used to generate logo message)
PRO900B80	Executable object file containing PR900B80.R, EIA80.R, KEYIN80.R and KIO80.R

PROLOG PROM
PROGRAMMER INTERFACE

Table A-4 lists the files contained on the 900B Z80 interface diskette.

TABLE A-4. 900B Z80 INTERFACE DISKETTE

Filename	Function
PR900BZ8.N	Prolog interface source file
PR900BZ8.O	Prolog interface source file
PR900BZ8.C	Command file used to edit, exchange, assemble and link PR900BZ8.N
PR900B80.R	Prolog interface relocatable file
EIAZ80.N	Source file of EIA port I/O routines
EIA80.R	Relocatable file of EIA port I/O routines
KIO80.R	Relocatable file of CRT routines
KEYIN80.R	Relocatable file of keyboard routines
MACLIB80.N	Macro library (used to generate logo message)
PR900B80	Executable object file containing PR900B80.R, EIA80.R, KEYIN80.R and KIO80.R

A.2 PROGRAMMING METHOD

The user may modify PR92080.N (920 8080/8085), PR920Z8.N (920 Z80), PR900B80.N (900B 8080/8085) or PR900BZ8.N (900B Z80) to allow interfacing with other PROM programming units. Each program transmits six bytes of address data (start address/end address), which is suitable for PROMs longer than 256 bytes. The beginning address is 0000 and the ending address is one less than the specified length. Data are sent to the PROM Programmer in hexadecimal ASCII, one nibble at a time. Therefore, the program must be modified for 4-bit PROMs.

After each byte is sent, the program pauses for approximately six milliseconds. After this pause, the program waits for the Clear To Send (CTS) line to go true.

The user should be thoroughly familiar with the operation of both the PROM Programmer unit and the personality module in use. (The 900B interface supports the optional 9115 module.) Refer to the appropriate Prolog manuals supplied with the PROM Programmer unit for detailed operating instructions.

A.3 CABLE CONNECTIONS

Each PROM Programming unit requires an RS-232/C cable with the pin connections shown in Table A-5.

TABLE A-5. PIN CONNECTIONS

Pin Number	Function
1	Chassis ground
2	Transmitted data (from ADS to Prolog)
3	Received data (from Prolog to ADS)
4	Request To Send (from ADS to Prolog)
5	Clear To Send (from Prolog to ADS)
7	Signal ground
20	Data Terminal Ready (from ADS to Prolog)

PROLOG PROM
PROGRAMMER INTERFACES

A.4 JUMPER CONNECTIONS

The jumper connections shown in Table A-6 are required on the ADS MPIO card (part #10155).

TABLE A-6. JUMPER CONNECTIONS

Jumper	Function
2	Connects transmitted data to pin 2
3	Connects Request To Send to pin 4
4	Connects Data Terminal Ready to Carrier Detect (internal to ADS)
6	Connects received data to pin 3
7	Connects Clear to Send to pin 5
9	Connects Data Terminal Ready to pin 20

The following jumpers must be removed from the MPIO card: 1, 5, 8 and 10.

A.5 LOAD INTERFACE PROGRAM

To load the interface program into the ADS, from the Debugger (JD) type:

L PR92080 (for 8080/8085 or Z80 920)
L PR900B80 (for 8080/8085 or Z80 900B)

The interface program begins at location 4000 hexadecimal.

INPUT DATA

I {xxxx }, {nnnn }

PURPOSE

The I command inputs data from the PROM in the PROM Programmer socket.

PARAMETERS

xxxx specifies the beginning address at which data will be stored.

nnnn specifies the number of bytes (hexadecimal) that will be transferred.

PROGRAM PROM

P {xxxx }, {nnnn }

PURPOSE

The P command programs the PROM with stored data.

PARAMETERS

xxxx specifies the beginning address from which data will be stored.

nnnn specifies the number of bytes (hexadecimal) that will be programmed.

PROLOG INTERFACE
LIST OF COMMANDS

RETURN CONTROL

R

PURPOSE

The R command returns control to the ADS Debugger (920) or to the 900B keyboard (900B).

NOTES

To resume execution of the PROLOG program, execute again beginning at location 4000. To resume transfers to and from the ADS, press the RETURN key.

B.1 INTRODUCTION

Four DATAIO PROM Programmer units are available for use with the ADS: System 7, System 9, System 17 and System 19.

NOTE

For Systems 7 and 17, ASCII HEX SPACE firmware is required.

The files contained on the DATAIO 8080/8085 interface diskette are shown in Table B-1.

TABLE B-1. DATAIO 8080/8085 INTERFACE DISKETTE

Filename	Function
DATAIO80.N	DATAIO interface source file
DATAIO80.O	DATAIO interface source file
DATAIO80.C	Command file used to edit, exchange, assemble and link DATAIO.N
DATAIO80.R	DATAIO interface relocatable file
EIA80.N	Source file of EIA port I/O routines
EIO80.R	Relocatable file of EIA port I/O routines
KIO80.R	Relocatable file of CRT routines
KEYIN80.R	Relocatable file of keyboard routines
MACLIB80.N	Macro library (used to generate logo message)
DATAIO80	Executable object file containing DATAIO80.R, EIA80.R, KEYIN80.R and KIO80.R

The files contained on the DATAIO Z80 interface diskette are shown in Table B-2.

DATAIO PROM
PROGRAMMER INTERFACE

TABLE B-2. DATAIO Z80 INTERFACE DISKETTE

Filename	Function
DATIOZ80.N	DATAIO interface source file
DATIOZ80.O	DATAIO interface source file
DATIOZ80.C	Command file used to edit, exchange, assemble and link DATIOZ80.N
DATAIO80.R	Relocatable DATAIO interface file
EIA80.N	Source file of EIA port I/O routines
EIA80.R	Relocatable file of EIA port I/O routines
KIO80.R	Relocatable file of CRT routines
KEYIN80.R	Relocatable file of keyboard routines
MACLIB80.N	Macro library (used to generate logo message)
DATAIO80	Executable object file containing DATAIO80.R, EIA80.R, KEYIN80.R and KIO80.R

B.2 PROGRAMMING METHOD

The user may modify DATIO80.N (8080/8085) or DATIOZ80.N (Z80) to accommodate other versions of the DATAIO PROM Programmer or to allow interfacing with other PROM programming units. The program transmits 50 nulls followed by an STX instruction and a \$A0000 instruction. Finally, the PROM data in ASCII hexadecimal are transmitted (separated by spaces). Transmission is terminated with an ETX instruction, followed by 50 nulls. Receive Data (from the DATAIO unit) is in the same format.

Checksum characters are not transmitted to the DATAIO unit. All characters other than the actual PROM data are stripped from the data stream prior to storage in RAM.

Since data transmission to and from the DATAIO unit involves commands to the DATAIO unit as well as ADS commands, the user should be thoroughly familiar with DATAIO operation. In general, a command set up on the sending device will not be executed until the corresponding command has been entered on the receiving device.

B.3 CABLE CONNECTIONS

The DATAIO unit requires an RS-232/C cable with the pin connections shown in Table B-3.

TABLE B-3. PIN CONNECTIONS

Pin	Function
2	Received data (from DATAIO to ADS)
3	Transmitted data (from ADS to DATAIO)
7	Signal ground

B.4 JUMPER CONNECTIONS

Table B-4 shows the jumper connections required on the MPIO card (part #10155).

TABLE B-4. JUMPER CONNECTIONS

Jumper	Function
1	Connects received data (from DATAIO) to pin 2
5	Connects transmitted data (from ADS) to pin 3
9	Connects DTR to DCD internally (permits receive)
10	Connects RTS to CTS internally (permits transmit)
11	Connects USART port 2 to EIA interface

The following jumpers must be removed from the MPIO card: 2, 3, 4, 6, 7 and 8.

B.5 LOAD INTERFACE PROGRAM

To load the interface program into the ADS, from the Debugger (JD) type:

L DATAIO80

The interface program begins at location 7000 hexadecimal.

INPUT DATA

I{xxxx},{nnnn}

PURPOSE

The I command inputs data from the PROM in the PROM Programmer socket.

PARAMETERS

xxxx specifies the beginning address at which data will be stored.

nnnn specifies the number of bytes (hexadecimal) that will be transferred.

PROGRAM PROM

P{xxxx},{nnnn}

PURPOSE

The P command programs the PROM with stored data.

PARAMETERS

xxxx specifies the beginning address from which data will be stored.

nnnn specifies the number of bytes (hexadecimal) that will be programmed.

DATAIO PROM
PROGRAMMER INTERFACE

RETURN CONTROL

R

PURPOSE

The R command returns control to the ADS Debugger.

NOTE

For additional information, refer to Application Note #020-1003 published by:

DATAIO
P. O. Box 308
Issaquah, WA 98027
(206) 455-3990
Telex 320290

SERIAL I/O PORT VALUES

Appendix C is currently being printed and will be shipped as soon as possible.

APPENDIX D
 AMERICAN STANDARD CODE FOR
 INFORMATION INTERCHANGE (ASCII)

TABLE D-1. ASCII CHARACTER CODE CONVERSION.

Dec	Octal	Hex	Function	Dec	Octal	Hex	Function	Dec	Octal	Hex	Function
0	000	00	NUL	43	053	2B	+	86	126	56	V
1	001	01	SOH	44	054	2C	,	87	127	57	W
2	002	02	STX	45	055	2D	-	88	130	58	X
3	003	03	ETX	46	056	2E	.	89	131	59	Y
4	004	04	EPT	47	057	2F	/	90	132	5A	Z
5	005	05	ENG	48	060	30	0	91	133	5B	[
6	006	06	ACK	49	061	31	1	92	134	5C	\
7	007	07	BEL	50	062	32	2	93	135	5D]
8	010	08	BS	51	063	33	3	94	136	5E	^
9	011	09	HT	52	064	34	4	95	137	5F	~
10	012	0A	LF	53	065	35	5	96	140	60	¯
11	013	0B	VT	54	066	36	6	97	141	61	a
12	014	0C	FF	55	067	37	7	98	142	62	b
13	015	0D	CR	56	070	38	8	99	143	63	c
14	016	0E	SO	57	071	39	9	100	144	64	d
15	107	0F	SI	58	072	3A	:	101	145	65	e
16	020	10	DLE	59	073	3B	;	102	146	66	f
17	021	11	DC1	60	074	3C	<	103	147	67	g
18	022	12	DC2	61	075	3D	=	104	150	68	h
19	023	13	DC3	62	076	3E	>	105	151	69	i
20	024	14	DC4	63	077	3F	?	106	152	6A	j
21	025	15	NAK	64	100	40	@	107	153	6B	k
22	026	17	SYN	65	101	41	A	108	154	6C	l
23	027	17	ETB	66	102	42	B	109	155	6D	m
24	030	18	CAN	67	103	43	C	110	156	6E	n
25	031	19	EM	68	104	44	D	111	157	6F	o
26	032	1A	SUB	69	105	45	E	112	160	70	p
27	033	1B	ESC	70	106	46	F	113	161	71	q
28	034	1C	FS	71	107	47	G	114	162	72	r
29	035	1D	GS	72	110	48	H	115	163	73	s
30	036	1E	RS	73	111	49	I	116	164	74	t
31	037	1F	US	74	112	4A	J	117	165	75	u
32	040	20	SP	75	113	4B	K	118	166	76	v
33	041	21	!	76	114	4C	L	119	167	77	w
34	942	22	"	77	115	4D	M	120	170	78	x
35	043	23	#	78	116	4E	N	121	171	79	y
36	044	24	\$	79	117	4F	O	122	172	7A	z
37	045	25	%	80	120	50	P	123	173	7B	{
38	046	26	&	81	121	51	Q	124	174	7C	
39	047	27	'	82	122	52	R	125	175	7D	}
40	050	28	(83	123	53	S	126	176	7E	~
41	051	29)	84	124	54	T	127	177	7F	DEL
42	052	2A	*	85	125	55	U				



5730 Buckingham Parkway
Culver City, CA 90230
(213) 641-7700/641-7200
TWX: 910-328-7202

May 27, 1980

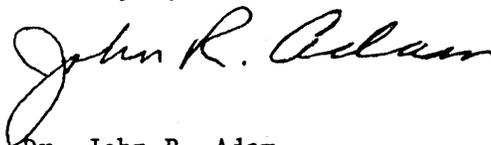
Dear Customer:

Enclosed is an update package for your 2300-ADS Software Reference Manual, manual #2300-5000-01. Vertical lines in the left and right margins mark the areas where changes have been made. The replacement pages should entirely replace the pages in your manual as listed below.

<u>Replace Page</u>	<u>With Page(s)</u>
1-5 and 1-6	1-5 and 1-6
1-7 and 1-8	1-7, 1-7A and 1-8
2-5 and 2-6	2-5 and 2-6
2-9 and 2-10	2-9 and 2-10
3-19 and 3-20	3-19, 3-19A and 3-20
4-5 and 4-6	4-5 and 4-6
6-1 and 6-2	6-1 and 6-2

Please call us with any questions or comments.

Thank you,



Dr. John R. Adam
Marketing Services Manager

JRA/ss

Commands that prompt for user verification during wild card operations are executed upon entry of

Y

for yes or

N, `RETURN`, `TAB`, or `STEP`

for no. `BREAK` aborts the operation.

Command Completion.

Unless otherwise specified, the `RETURN` key must be pressed at the end of a command line in order for the command to be executed.

1.5 FILENAME CONVENTIONS

In the syntax descriptions that follow, the word "filename" refers to a filename without prefixes or parameters. The word "file-spec" refers to a filename including any optional prefixes or parameters (see Sections 1.6 and 1.7 for information on prefixes and parameters).

ADS system programs accept only filenames conforming to the following conventions.

- 1) A filename must be from one to ten characters in length.
- 2) The first character must be a letter. Subsequent characters may be letters, digits or periods (see Exceptions on page 1-6).
- 3) Spaces are not allowed as filename characters; if inserted, they are ignored.

Several example filenames are listed below:

TESTPROG	- Valid
FILE29	- Valid
DRIVER3.R	- Valid
BIG.PROGRAM	- Invalid; longer than ten characters

INTRODUCTION

- | | |
|----------|--|
| 9900PROG | - Invalid; first character is not a letter |
| MY-FILE | - Invalid; hyphen is an illegal character |

Exceptions.

Characters other than letters, numbers and periods may appear in a filename if the filename is enclosed in quotation marks. Also, if enclosed in quotes, a filename may begin with a non-alphabetic character. For example, the two invalid names, 9900PROG and MY-FILE can be validly specified as:

```
"9900PROG"  
"MY-FILE"
```

A filename may legally contain a quotation mark if it is followed by a second quotation mark and if the entire filename is enclosed in quotes. For example:

```
"ABC""DEF"
```

specifies file ABC"DEF.

1.6 FILENAME PREFIXES

Three types of prefixes may optionally precede a filename. Any two types or all three types may be used in the same filename specification, provided they are entered in the following order:

```
[new/old-spec:][device-spec:][(attribute-spec)]
```

- 1) New/Old Prefix. The ADS assumes that when an output filename is called for, the existing file should be overwritten or, if it does not exist, a new file should be created (Manager only). In some instances the user may wish to override this default. Two prefixes are available for this purpose.

- a) N: Prefix. The N: prefix instructs the ADS to create a new file, but not to write over the file if it exists. Examples:

```
N:OUTFILE      - specifies a new file on drive 0 named  
                OUTFILE;
```

```
N:1:DEVTABLE   - specifies a new file on drive 1 named  
                DEVTABLE;
```

- b) 0: Prefix. The 0: prefix instructs the ADS to write over an existing file and insures that an error will result if the file does not exist. Examples:

0:OUTFILE - specifies an existing file on drive 0 named OUTFILE;

0:1:(S)TEXT2 - specifies an existing source file on drive 1 named TEXT2.

NOTE

New filenames specified to ADS system programs other than the Manager must contain the N: prefix. 0: is the default and is not necessary when specifying existing filenames.

- 2) Device Prefix. A device prefix specifies the physical device to or from which a file will be transferred. The various disk drives, the printer and the serial output ports are treated as file-oriented devices by the ADS. Thus, the user may select any available device by entering a device prefix with a filename. If a device prefix is not entered, the ADS defaults to disk drive 0.

- a) Disk Drive. A disk drive is specified by prefixing the filename with the desired drive number followed by a colon. For example:

1:CMDPROG
0:TABLE.Z80

- b) Parallel Port Printer. Any output file may be transferred directly to the printer by entering:

P:

P: is the entire filename specification. For example:

M PROG,P:

transfers the file PROG to the printer. Refer to the List of Commands at the end of Section 2 for more information on the M command.

- c) Serial Port. A serial port is specified as:

S[n]:

where n is the optional port number. The default port number, as well as the baud rate and parity options may be preset using the Manager's I command (see List of Commands at the end of Section 2). Presettings may be overridden by including parameters in the filename specification (see Section 1.7).

- 3) Attribute Prefix. The Manager program permits the user to specify the attributes associated with a file. (These specifications are one-letter codes which identify the file type. Refer to Section 2.2 for additional information on file attributes.) When it is necessary to specify an attribute(s) with a filename, enclose the attribute(s) in parentheses. For example:

(S)NEWFILE	- indicates a source file named NEWFILE;
(PO)MASTER	- indicates a permanent object file named MASTER;
1:(R)SUBR2	- indicates a relocatable file named SUBR2.

1.7 FILENAME PARAMETERS

In some instances, additional information (other than the filename itself and optional prefixes) is required for a filename specification. For example, when a serial port file is specified, parameters may be appended to specify the baud rate and parity options. A filename parameter is of the form:

```
{file-spec}[/parameter name=parameter value]
```

Two types of parameter values may be used.

- 1) Integer Parameter Value. An integer parameter value is used when a decimal number must be specified. For example, when a serial output port is specified, the baud rate may be specified as well:

```
I S2:/BAUD=1200
```

In the example above, serial port 2 is initialized to 1200 baud. Refer to Appendix C for additional information on baud rates.

- 2) String Parameter Values. String parameter values are similar to integer parameter values except that the value following the equal sign may be any string of characters. If characters other than letters, numbers or periods are included, the string must be enclosed in quotation marks. A quotation mark may appear in a string parameter if it is followed by a second quotation mark.

String parameters are necessary when specifying serial port parity. For example:

```
/PARITY=ODD
```

Single, Multiple or No Character Wild Card Symbol.

The asterisk (*) character within a filename implies any group of zero or more characters. For example, using the Move command:

```
M*,1:*
```

moves all files (except those whose filenames are enclosed in quotes) from drive 0 to drive 1. The command:

```
M*,1:"*"
```

moves all files (including those whose filenames are enclosed in quotes) from drive 0 to drive 1.

To scratch (delete) a collection of files from drive 0 whose filenames begin with the characters SOURCE, the command:

```
S SOURCE*
```

is used. The files SOURCE.0, SOURCE.N and SOURCE123 are deleted, but the files RESOURCE and ASOURCE1 are not affected.

To assign attributes to a collection of files on drive 0 containing the string GO, the command:

```
A*GO*/A=S
```

is used. Every file whose name contains the string GO (such as AGONY, GOSH or AGO) is assigned the S attribute.

Wild Carding by Attribute.

Files may be wild carded by attribute, as well as by name, or by a combination of the two. The attribute specification, shown in parentheses, must follow the drive number and colon, if present. For example:

```
M (S)*,1:*
```

moves all source files from drive 0 to drive 1.

If multiple attributes are coded, only those files having all of the specified attributes are processed. For example:

```
S (OW)*
```

scratches only those files with both the O and W attributes. Files with only one of these attributes are not scratched.

Manager Queries During Wild Card Operations.

When the Manager prompts for verification before executing a command, the user may respond with Y (the file is processed) or N, [STEP], [TAB] or [RETURN] (the file is skipped). After receiving verification from the user, the Manager will continue to prompt for subsequent files. To abort the entire operation, the [BREAK] key is pressed.

When the Manager prompts for information such as attributes during an A command, the [TAB] key or [STEP] key may be used to skip the file and leave its attributes unchanged. Entering a [RETURN] clears all attributes from the file.

2.7 OPTION SWITCHES

Many Manager commands may be appended with an option switch. Five switches are available. Each switch must begin with a slash. Refer to the List of Commands at the end of Section 2 for specific uses of the option switches as they apply to individual commands.

/A Switch.

Some commands ask for user verification before executing the commanded function. The /A option switch may be used to override this verification. When the /A switch is specified, all files are processed as if Y were typed in response to each query.

/C Switch.

If the number of files processed exceeds the space available on the display screen, the Manager will pause and the message "PRESS ANY KEY TO CONTINUE" will appear. The /C option switch suppresses this message as well as all queries and prompts (the Manager assumes Y). The /C switch is particularly helpful when using command files and the length of displayed information is unknown.

/Q/ Switch.

The /Q switch requests verification for every case. Thus, the user may select a subset of the wild carded files by typing Y for yes, N, [STEP], [TAB] or [RETURN] to skip to the next case, or [BREAK] to abort the operation.

/P Switch.

The /P switch routes all data on the display screen to the Centronix compatible printer.

ASSIGN ATTRIBUTES

```
A{file-spec}{[/P]}/A=attribute }
A{wild-spec}{[/P]
  [ /A ]
  [ /C ]
  [ /Q ]
}[/A=attribute]
```

PURPOSE

Attributes help identify files and protect them against accidental destruction.

PARAMETERS

/A assigns new attributes and overwrites any existing attributes. Any of the following attributes may be specified:

- O executable absolute object file
- P permanent file
- R relocatable object file
- S source file
- W write-protected file
- Z permanent, write-protected file

NOTES

A file may be assigned any combination of attributes. Attributes may be added and removed to accommodate ADS system program requirements (see Section 2.2, Table 2-A).

The wild card feature may be used to change the attributes of a group of files. If the attributes parameter is omitted, the Manager will prompt for the attributes for each file individually. The **TAB** key may be used to leave the attributes of any file unchanged and skip to the next file.

See Section 2.7 for option switch definitions. All desired option switches must be specified before the /A=attribute parameter is entered.

MANAGER

LIST OF COMMANDS

CREATE FILE

C{filename}[/I=initial-alloc[/E=extension-alloc]][/P][C]

PURPOSE

The C command creates a new file and allocates for that file the number of tracks specified in the /I parameter.

PARAMETERS

/I specifies the initial size of the file. When data are written into the file and the file exceeds its initial allocation, additional space is automatically allocated until the disk becomes full.

/E specifies the number of additional tracks to be allocated when the file expands beyond its allocated size. If this parameter is omitted, the default is one track at a time (each time the file expands beyond its allocated size).

NOTES

As stated in Section 1.6, new files are created automatically. The C command provides the additional feature of allowing a track allocation to be specified.

The C command does not support the wild card feature; only one file may be created at a time.

See Section 2.7 for option switch definitions.

END PREVIOUS FILE?

An attempt was made to write to a new output file without ending use of the current output file (E command). To end the previous file, type Y; to cancel the write command, type N or RETURN.

END FILE

The Editor has reached the end of the input file.

FILE NOT SPECIFIED

An L, W, N, G, CT or CF command was entered without specifying a filename, and input or output files were not previously specified. Retype the L or W command and include the desired filename, or execute a W command with a filename before retyping the N or G command.

FULL - MORE DATA IN FILE

The work space was filled with data from a Load operation. Before the remaining data can be loaded, a portion or all of the work space data must be written out.

NOT FOUND

During an F or G command operation, a character string match was not found in the range of search.

NOT A SOURCE FILE

The file designated in an L, W, CT or CF command does not have an S attribute. Use the Manager's A command to change the file's attribute(s).

OVERFLOW

The work space is full. All or part of the work space should be written to the output file using an N or W command.

REPLACE THIS LINE?

The F or G command requires verification before replacing a string in the line displayed. Type Y to replace, N, RETURN, TAB or STEP to continue without replacement, or BREAK to cancel execution of the command.

SAVE DATA?

An attempt was made to jump to another ADS system program without writing the work space contents to an output file. Type N to erase the text in the work space and allow the program jump. To cancel the jump command and preserve the work space text, press any key other than N.

SYNTAX

There was a syntax error in the last command entered. Retype the command.

4.5 SPECIFYING ASSEMBLER FILES

Refer to Figure 4-4. After the desired options have been selected and entered, the Assembler will generate a series of prompts, as listed below. The user may respond to each with any legal file specification as shown in Figure 4-4 (see Sections 1.5, 1.6 and 1.7 for allowable filenames, prefixes and parameters).

- 1) SOURCE FILE - Any filename entered must have the S attribute.
- 2) MACRO FILE - This input is optional (enter RETURN to bypass). See Section 4.10 for additional information on MACRO files.
- 3) OUTPUT FILE - This input is optional (enter RETURN to bypass). Any filename entered must have the R attribute (a new file created by the Assembler will automatically be assigned the R attribute).
- 4) LISTING FILE - This input is optional (enter RETURN to bypass). Any filename entered must have the S attribute or none at all.

Figure 4-4. Example Assembler Display.

ASSEMBLER

4.6 ASSEMBLER PROCESSING

After the last filename has been entered and the `[RETURN]` key has been pressed, assembly begins. Statements are processed in two passes. On the first pass the Assembler assigns values to symbolic labels and stores them in a symbol table. During pass 2 the Assembler analyzes each statement from the input file and generates relocatable object output. If the Assembler recognizes an error, the statement is flagged and zero-bytes are generated for that statement.

4.7 HALT ASSEMBLY

To halt the assembly process at any point, press the `[RETURN]` key. This is useful for viewing the Assembler output as it scrolls on the display. The Assembler will pause and prompt:

CONTINUE?

To continue, type:

Y or `[RETURN]`

To abort assembly, type:

N or `[BREAK]`

4.8 ASSEMBLER STATEMENT SYNTAX

Statements are composed of four components or fields. Fields are separated by a space or a series of spaces. Statements are entered one per line with a maximum of 80 characters.

Label Field.

The optional label field may contain a symbolic name which is used to reference the statement or which will be assigned a value by an EQU directive. Labels must begin in the first character position in the line. Labels may be from one to eight characters in length. The first character must be a letter (A-Z). Subsequent characters must be letters, dollar signs or numbers (0-9).

During assembly a value is assigned to each label, as described below.

- 1) ASEGs. In ASEGs, the value assigned to a label is the absolute address at which the assembled statement will be stored. This value is identical to the value in the absolute segment location counter, which appears on the left side of the assembly listing.

6.1 INTRODUCTION

The Command File Processor reads command lines from a diskette file and passes them to ADS programs, thus providing a convenient alternative to the time-consuming method of keyboard command entry. The Command File Processor is particularly helpful when long sequences of commands must be entered repeatedly. Command files may reside on a diskette in any disk drive.

6.2 COMMAND FILE CREATION

A command file consists of a number of legal ADS commands (one command per line). Command files may be created with the Editor program. The Editor automatically assigns the S attribute to newly created files (the Editor will work only with S-attributed files).

6.3 INITIATE COMMAND FILE PROCESSING

To transfer control from the keyboard to the Command File Processor, type the following sequence:

```
J^{filename}[,parameter-1][,parameter-2]...[parameter-9]
```

filename is the name of the source (S attribute) file in which commands have been stored; the Command File Processor reads commands from this file. parameter-1 is a string of characters which directly replaces the ^1 sequence in the command file; parameter-2 replaces ^2, etc. (a maximum of 9 parameters may be entered). The number of parameters entered at the initiation of command file processing must be equal to or greater than the highest numbered substitution sequence ^n in the command file (see Section 6.4).

Two successive commas (,,) in the initiation command denote a null parameter and cause every occurrence of the corresponding substitution sequence ^n to be deleted and the part of the line to the right of ^n to be moved left two character positions. If the last parameter in an initiation command is null, an extra comma must be entered at the end of the command.

In the following example, four parameters have been entered. The first and last parameters are null. The substitution sequence in the command file (FILEA) must be no higher than ^4.

```
J^FILEA,,FILEX,P,,
```

In some ADS system programs an S option invokes symbol table generation. For instance, if the command file contains:

```
JA      (start Assembler operation)
TL^3    (Assembler options)
```

the third parameter of the initiation command `J^FILEA,,FILEX,S,,` causes the options `TLS` to be entered. If the third parameter is null, the options `TL` are entered. Thus, the user may choose to assemble with symbol table output at the initiation of command file processing rather than at the time the command file is written.

NOTE

If the Assembler or Linker is used in a command file, a blank line is required at the end of the input sequence to continue command file processing.

6.4 COMMAND FILE PROCESSOR FEATURES

Four features are provided by the Command File Processor, as described below.

^L Feature.

The ^L keyboard escape feature is entered as:

```
[...part of command line]^L[part of command line...]
```

This feature allows characters to be entered from the keyboard until the `[RETURN]` key is pressed.

Parts of the command file line are combined with characters entered at the keyboard to form a complete command line. The first part of the command line is read from the command file. Input is then taken from the keyboard. When the `[RETURN]` key is pressed, the Processor reads the next part of the command from the command file. The keyboard-entered characters are inserted in the command at the ^L position. The `[RETURN]`, which ends keyboard input, is not inserted.

If ^L appears by itself in a command file line, the Processor reads only keyboard-entered characters for that line.

In the example below, the Editor's Find command is constructed from parts of a command file line. The characters ^L are replaced with the keyboard-entered characters `S20` during execution.

line in command file:

```
F,R2,^L,AV
```

keyboard entry for ^L:

```
S20 [RETURN]
```

effective command input to the Editor:

```
F,R2,S20,AV [RETURN]
```