PART 4

THE BATCH USER'S GUIDE

PART 4

CHAPTER 1

HOW TO USE BATCH

1.1 INTRODUCTION

This Chapter contains the basic information required to prepare a job for execution by PDP-11 Batch operating system. While the input medium is assumed to be the card reader in the following examples, Batch supports a variety of input devices: disk, DECtape, magnetic tape, cassette or paper tape (refer to Table 4-2).

Batch includes a disk-resident Monitor, and a number of system programs (such as the FORTRAN Compiler). The Monitor controls execution of user jobs, by reading and interpreting the batch stream, batch command statements the user has placed in the input deck. Several jobs can be processed sequentially by Batch, each job set apart from its neighbors by delimiters that define its starting and ending points. The sample job shown below consists of a FORTRAN program to be compiled, linked, and executed. See Figure 4-1.

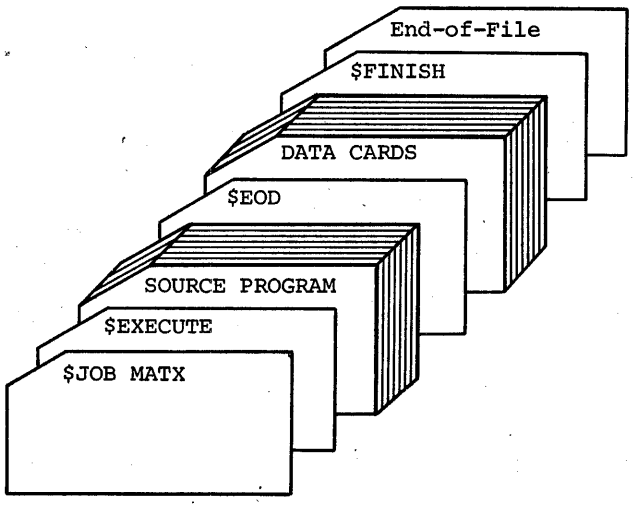


Figure 4-1
Sample Batch Job

When the $EXECUTE command is used, the Batch system determines the input and output specifications for the FORTRAN Compiler. Thus, in the sample deck shown in Figure 4-1, the object program is output to the system device, the source listing is output to the line printer, and the source program is read in from the batch stream. The user can include input/output specifiers with the $EXECUTE statement, or he can use the $RUN command instead of $EXECUTE, followed by a command string. The command string consists of the input/output specifications, and is distinguished by having a # character in colum 1. Figure 4-2 shows a typical command string. Note the "/GO" at the end of the # card - this is a switch that causes the FORTRAN program to be compiled, linked, and executed, just as though the $EXECUTE command had been used.

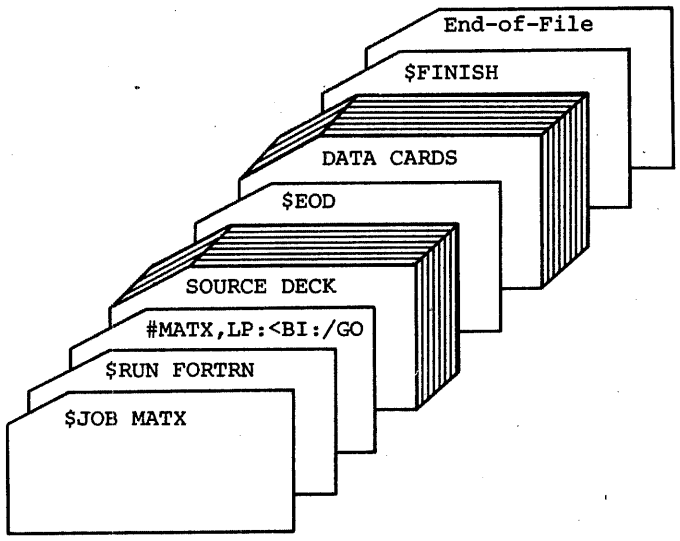


Figure 4-2
Use of /GO Switch

In Figure 4-2, the output is specified as:

MATX - object program (file name MATX; the extension .OBJ is assigned by default);

LP: - the line printer will be used to list the source program.

The input is:

BI: - the source program comes from the batch stream (BI);

/GO - the /GO switch causes the program to be compiled, linked, and executed.

The user also has the option of specifying each step of a job's processing. For example, he may wish to have the system generate a dump in the event of a fatal error in his program. To do so, he must include Batch Command Language cards at each step of the job. As shown in Figure 4-3, the dump is specified on the $RUN MATX/DU card, by the switch, /DU.

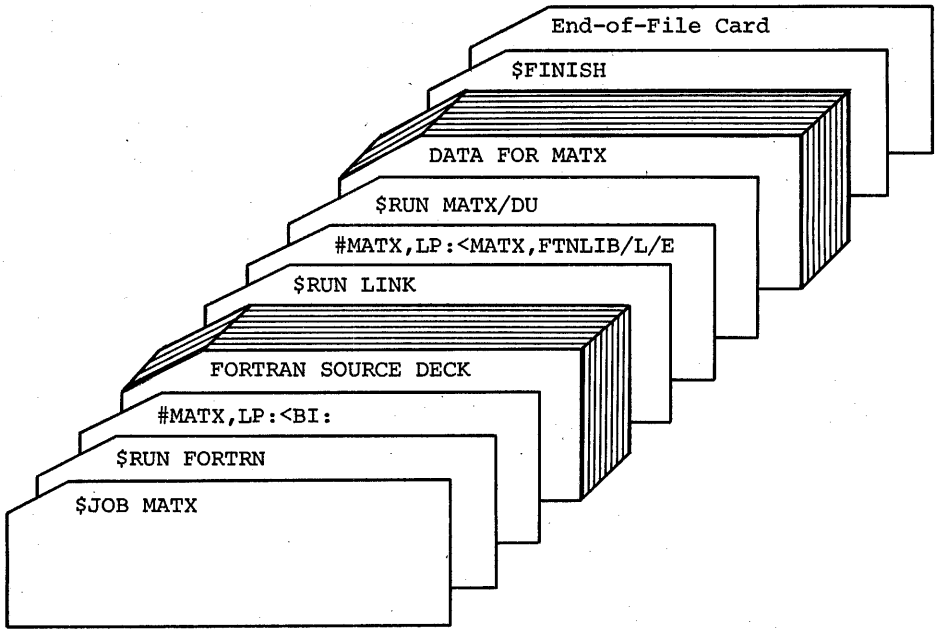


Figure 4-3
Batch Job Set-Up with User-Specified Job Steps

Table 4-1 defines the function of each BCL card in the deck.

TABLE 4-1

Key to Card Deck in Figure 4-3

$JOB MATX	Defines the starting point of the job and gives the job name, MATX.
$RUN FORTRN	Causes the FORTRAN Compiler to be loaded and executed.
#MATX,LP:<BI:	Command string defining the FORTRAN Compiler's input and output datasets. Note that the GO switch is not used.
FORTRAN source deck	The cards comprising the source program, MATX.
$RUN LINK	Causes the Link-11 Linker to be loaded and executed.
#MATX,LP:<MATX/CC,FTNLIB/L/E (See note)	Defines the Link-11 load module to be MATX; the Map output to be on the line printer; and the object modules to be MATX (the output of the FORTRAN Compiler) and FTNLIB, library routines (as indicated by the /L switch). The /E switch signifies end of Linker input.
$RUN MATX/DU	Causes the linked program, MATX, to be loaded and executed. The /DU is a "DUMP" switch, requesting that the program be dumped if an error occurs.
DATA FOR MATX	Data to be used during program execution.
$FINISH	The $FINISH card is the end-of-job delimiter; it designates the logical end of the job.
End-of-File	This card physically delimits the job. The operator must place one of these cards at the end of each job. As a further safeguard, the user should put one as the first card in his deck, to protect himself from a preceding job's erroneous execution. Eight consecutive end-of-file cards terminate the batch stream.

NOTE

The /CC switch (concatenate) must be used if the source deck comprises more than one main program or subroutine in a concatenated deck. The /CC tells the Linker that the object module contains concatenated modules.

1.2 STANDARD BATCH PERIPHERAL DEVICES

As discussed in preceding chapters, standard DEC peripheral devices must be referred to by specific mnemonics. In addition to those previously mentioned two new devices are introduced here, the pseudo devices. See Table 4-2.

TABLE 4-2

Standard Batch Peripheral Devices

Name	Mnemonic
Disk	DP DF DK DC
DECtape	DT
Line Printer	LP
Magtape	MT
Cassette Tape	CT
Paper Tape Punch (High-speed)	PP
Paper Tape Reader (High-speed)	PR
Low-speed Punch and Reader	PT
Keyboard	KB
Card Reader	CR
Batch Input	BI*
System Device	SY*

*This is a pseudo-device. Refer to Section 4-1.2.1.

1.2.1 Use of Pseudo-Device Specifiers (BI, SY)

By specifying BI as the input device, the user achieves device-independence; the source program is read from the batch input device, regardless of what the device may be. This feature of Batch permits the same control card to be used without concern for which device the batch stream may be read from.

The pseudo-device BI must be used whenever the batch stream is coming from cards or papertape. Batch stream inputs from multi-dataset devices such as disk can use the explicit device mnemonic. However, BI should be used for device independence.

To specify that the system device is to be used, when the actual device is not known, specify SY. The system will supply the correct device for SY.

1.2.2 Device Assignment

1.2.2.1 FORTRAN Logical Units

The BI pseudo-device specifier is assigned to logical unit 8 in the FORTRAN device table.

Example:

```
READ (8,23) A,B,C (read from BI)
```

The $ASSIGN command can be used to override the default values.

```
$ASSIGN BI:,4
```

Logical unit 4 is assigned to the batch stream dataset.

1.2.2.2 Macro Device Assignment

It is possible in Macro to access the BI pseudo device via direct reference to BI:,

Example:

```
        .READ  #LNKBLK, #FILBLK    ;read from BI

        .WORD ERROR  *             ;ERROR RETURN ADDRESS
LNKBLK: .WORD Ø
        .RAD5Ø/IN/
        .WORD 1
        .RAD5Ø/BI/
```

1.3 BATCH OPERATING PROCEDURES

1.3.1 Entering Batch Mode

Two basic procedures are involved in getting started with DOS/BATCH. The first procedure, loading the Monitor, is accomplished through the console. The second procedure, entering Batch mode, is done via the keyboard.

When the Monitor has been loaded into core, it responds by printing

```
DOS/BATCH Vxx-xx
DATE:
```

at the terminal. The date must be supplied by the user in the form

```
dd-mmm-yy
```

where

dd	is the current day.
mmm	is the first three letters of the month.
yy	is the last two digits of the year.

Any other response will yield a WRONG DATE! message from the Monitor. After the date is supplied, the Monitor responds by printing

```
TIME:
```

to which it expects the current time given in the format of hours:minutes:seconds (hh:mm:ss). The Monitor then prints a $ to indicate readiness for a user command. At this point, the user logs in to the system by typing LO[GIN] [uic], where uic is his user identification code, as described in Section 3.2.4.

When the Monitor responds by printing $, the appropriate response to invoke Batch mode is to type the following command string immediately after the $, and on the same line.

```
$BA dataset1[/switch(es)] [,dataset2]
```

where

dataset1 is the batch stream dataset
dataset2 is the default log dataset[1]

The switches that may be specified are the time-limit (/TI) switch, no-echo switch (/NE), and default-log (/LO) switch. The time limit switch governs the maximum duration of a job. It is specified as

/TI:hh:mm

where hh and mm are specified as decimal digits. If both are included, hh equals hours, and mm equals minutes. If only one value is given, it is assumed to be minutes.

The no-echo switch suppresses printing of commands at the keyboard. It is specified as /NE. If this switch is included, only the $JOB command is echoed at the keyboard with the start and finish times of the jobs.

The default-log switch specifies that the default log dataset is to be used. It is specified as /LO. If this switch is specified, it causes log information to be placed on a system file on the disk (CMO.SYS) in the user's area. When the $FINISH card is read, this file is automatically output by the system, in accordance with the appropriate cleanup file (as described in the System Manager's Guide). Thus, the advantage of specifying /LO is that log output is collected and output all together, rather than being interspersed among unrelated outputs on the listing. Note that when /LO is specified, a log dataset (dataset 2) must not be specified with the $BATCH command, since this creates a conflict. If the /NL switch has been specified on the $JOB card, log processing is inhibited for that job.

A no file error message at the keyboard indicates that the batch stream could not be found. The system searches first the current user's area; if it fails to find the batch stream there, it searches the system area. If the batch stream is not found there, a no file message results.

The time-limit switch given with this BATCH command takes precedence over any specified in a $JOB command.

---

[1]Output related to syntax errors, is printed on the teleprinter, if the log dataset is omitted.

Sample BATCH Command

```
$BA CR:/TI:2Ø,LP:
```

The dataset specifiers in the example designate the batch stream device to be the card reader, with the log produced on the line printer. The default time limit is set to 2Ø minutes for each $JOB in the above example.

1.3.2 Operator-System Communication

Batch mode provides several ways for the operator and the system to communicate with each other: the Monitor may print information regarding the status of a job (e.g., error messages); a user program may request operator action; or, the operator may wish to exercise control of system operation, or respond to a system request.

1.3.2.1 Error Messages

The actions taken as the result of an error in Batch mode are described below. The messages produced are summarized in Appendix K of this handbook.

Action messages (Annn) are printed on the terminal, but do not appear in the log returned to the user. The system suspends operation until the operator responds at the keyboard. Other classes of messages (I, S, W, and F) are printed at the teleprinter, and in the log (unless the user has suppressed the log).

If a system program error occurs, subsequent input from the batch stream is ignored until detected, or a Monitor command is read.

A fatal (F) error causes the job to be aborted. If the user has specified dump-on-error, he is given an octal listing of the contents of the area he specified to be dumped.

1.3.2.2 Messages to the Operator

Messages can be sent to the operator from the batch stream by means of the $ME command, which is formatted here.

$$\$ME \left\{ \begin{matrix} \Delta \\ , \end{matrix} \right\} text$$

The $ME command adheres to the syntax conventions used by the other Batch mode commands; i.e., the $ must be in column 1. The command itself is separated from the message by either a comma or a space, as indicated by $\left\{ \begin{matrix} \Delta \\ , \end{matrix} \right\}$.

The program will continue execution, following the issuance of $ME text, unless the user issues a $WAIT command; in this case, the operator is required to type in CO to effect resumption of the program.

Example:

Entries supplied by user in batch stream (assume job name to be MATX):

```
$ME MOUNT TAPE XYZ ON UNIT 1
$ME DO NOT WRITE ENABLE
$WAIT
```

Response printed at keyboard (as seen by the operator):

```
MATX    :MOUNT TAPE XYZ ON UNIT 1
MATX    :DO NOT WRITE ENABLE
AØ5Ø    ØØØØØØ (action message indicating that $WAIT statement is in
                effect.)

$
```

The operator types CO (followed by the RETURN key) on the $ line to resume the program, after the action has been taken.

1.3.2.3 Operator Commands

The operator notifies the Monitor of his intention to type in a command by pressing the CTRL and C keys simultaneously. (This action is indicated as CTRL/C.) The Monitor then responds by printing a period at the start of the next line. The operator then types the appropriate command.

To abort the current job, the operator/Monitor message sequence is

```
CTRL/C
.KI         (the period (.) is printed at the keyboard in
            response to CTRL/C)
```

To terminate the batch stream, the operator types

```
CTRL/C
.TE
```

after which the system leaves Batch mode, returning the Monitor to keyboard mode. TE is legal only from the keyboard. If entered through the batch stream, the current job is aborted with the INV CMD! message.

1.3.2.4 Commands Printed at the Keyboard

The batch system prints some commands at the keyboard to help the operator monitor a job's progress. For example,

```
$RUN FORTRN
```

is printed at the keyboard when this card is read. (It is also output to the job log.) $JOB, $GET, $CHANGE and $FINISH are also printed. (If the /NE switch is used, only $JOB is printed.)

PART 4

CHAPTER 2

BATCH COMMAND LANGUAGE

2.1 BATCH COMMAND LANGUAGE

The user communicates with the Batch Monitor through Batch Command Language (BCL) statements. For example, to prepare a FORTRAN job for execution, he must include statements to:

a. Define and delimit the job;

b. Effect compilation of source code;

c. Link object modules; and,

d. Execute the program.

Batch Command Language includes most of the DOS monitor keyboard commands, the special Batch commands (e.g., $JOB), and the concise commands. It is assumed that the reader is familiar with the DOS/BATCH Monitor.

Batch Command Language statements directed to the Monitor, must observe the following rules:

1. A dollar sign ($) must appear in column 1;

2. The statement identifier must immediately follow the $, starting in column 2.

3. The statement identifier is terminated by a comma or a space. Therefore, neither of these characters can appear as part of the identifier.

2.1.1 Batch Commands

These include most of the DOS commands and the special batch commands:

1. $CHANGE

2. $EOD

3. $JOB

4. $ME

5. $OWN

which are only valid as input from a batch stream.

The effect of a DOS command in Batch mode depends on whether the command is received from the keyboard (following CTRL/C) or from the batch stream. Table 4-4 lists all commands and their status in Batch mode.

Table 4-4

Batch Commands

Command	Function	
	From Keyboard	From Batch Stream
$ASSIGN	Assign a physical device and filename to a dataset.	Same as from keyboard. See Note 1.
$BATCH	Invokes batch mode.	Invalid.
$BEGIN	Invalid.	Honored only when program loaded, and never started.
$CHANGE	Invalid.	Transfers batch stream to dataset specified.
$CONTINUE	Resumes program execution.	Ignored.
$DATE	As in interactive mode.	Cannot contain a value. Causes date to be printed in the log.
$DUMP	Dumps core to line printer. Processing suspended until dump complete.	Same as from keyboard.
$ECHO	As in interactive mode.	Invalid.
$END	As in interactive mode.	Invalid.
$EOD	Invalid.	Delimits physically contiguous, logically distinct data.
$FINISH	Invalid.	Logical job delimiter.
$GET	Invalid.	Loads a program from the specified device.
$JOB	Invalid.	Logical job delimiter.

(continued on next page)

Table 4-4 (Cont.)

Batch Commands

Command	Function	
	From Keyboard	From Batch Stream
$KILL	Terminates the current job.	Terminates the current program. See Note 1.
$MESSAGE	Invalid.	Outputs a message to the operator.
$MODIFY	As in interactive mode.	No display of location's previous contents.
$ODT	Invalid.	Invalid.
$OWN	Invalid.	Allows unformatted reads from batch stream.
$PRINT	As in interactive mode.	Invalid.
$RESTART	Invalid.	Restarts a program.
$RUN	Invalid.	Loads and starts a program.
$SAVE	Invalid.	Writes the program in core, onto the disk in loader format.
$STOP	Invalid.	Invalid.
$TERMINATE	Terminates the batch session.	Invalid.
$TIME	Enter a value for TIME or display the time.	Request that TIME's value be output to the log. Entering time is illegal.
$WAIT	As in interactive mode.	Suspends job execution until CO is typed at keyboard.

Note

1. Other commands that force a KILL, if read when the program is reading command string input, are: $RUN, $GET, $FINISH, $CHANGE, and $JOB.

2.1.2 Concise Commands

The $EXECUTE command, discussed, is one of the BCL commands referred to as "concise commands". They are called concise commands because they allow the user to invoke whole sequences of commonly-used functions with a single command, instead of two or more otherwise required. Concise commands are summarized in Table 4-5.

Table 4-5
Batch Concise Commands

NAME	FUNCTION
$CPY	Copies a file, or files onto a specified output dataset.
$DEL	Deletes specified datasets.
$DIR	Obtains a directory listing.
$EX[ECUTE]	Causes a source program to be compiled (or assembled), linked, and run.
$FORTRN	Compiles a source program, producing an object module and/or listing.
$LINK	Links object modules into an executable load module, and generates a load map.
$LIST	Prints datasets on the line printer.
$MACRO	Assembles a source program into an object module, and produces a listing; or, if specified, produces only a listing.
$RNM	Renames a dataset.

2.1.3 Synchronous/Asynchronous Commands

Certain commands are treated as asynchronous commands, while the rest are dealt with synchronously. Asynchronous commands cause action to be taken as soon as they are read, regardless of whether they are read from the Batch command input dataset (logical name CMI); from the system program command input dataset (PCI)[1]; or from a user's dataset.

[1]Command datasets are discussed in Chapter 4-4.

Asynchronous commands:

$BEGIN
$DATE
$DUMP
$EOD
$KILL
$MESSAGE
$MODIFY
$OWN
$RESTART
$SAVE
$TIME
$WAIT

Synchronous commands are those which are not executed immediately unless they are read while the program is reading CMI. If read from PCI, or a user's dataset, they cause an "end-of-file" to be returned to the program; the command is held until a READ CMI is issued, at which point the command is executed.

Synchronous commands:

$ASSIGN
$CHANGE
$FINISH
$GET
$JOB
$RUN

All synchronous commands, except $ASSIGN, force a $KILL. $JOB also forces a $FINISH.

2.1.4 Monitor Command Statements

Commands are arranged in alphabetic order. Only those commands that are valid or effective from the batch stream are discussed (refer to Tables 4-4 and 4-5).

ASSIGN

2.1.4.1 $ASSIGN

Format:

$AS[SIGN]{' Δ}[dataset specifier, logical name]

Purpose:

This command assigns a physical device (and a file name, when the device is file-structured) to the dataset identifier by logical name. The format of dataset specifier is

```
dev:filename.ext[uic]
```

where dev designates the device, and filename.ext[uic] designates the name, extension, and uic, if any, to be assigned to the file.

The logical name is the name that has been specified in the link block in the user's program.

NOTE

> The $ASSIGN command should not be used with Batch command datasets (i.e., CMO, CMI, PCI, and CDI). These datasets are used for input to the Command String Interpreter (CMI), related output (CMO), input to system programs (PCI), and data input resulting from a program command (CDI). Refer to Chapter 4-4 for details.

The duration of an $ASSIGN depends on when it was issued. If issued at the job level (i.e., after $JOB, but prior to $RUN or $EXECUTE), an $ASSIGN remains in effect for the duration of the job, unless subsequently altered. If issued at the program level, an $ASSIGN is in effect for the duration of that program, unless changed during execution of the program. An $ASSIGN with no arguments releases (deassigns) all assignments previously made by the current job.

Examples:

To assign a DECtape file named COM.BN to the dataset with the logical name ITR:

```
$AS DT:COM.BN,ITR
```

To assign a disk file to FORTRAN unit number 5:

```
$AS FILE.EXT,5
```

BEGIN 2.1.4.2 $BEGIN

Format:

$BE[GIN] [{,/Δ} address]

Purpose:

The $BEGIN command starts execution of an already loaded program at the stated address. The address value, if specified, is an absolute octal value; if not stated, the normal start address is used.

The $BEGIN command is used only for programs that have been loaded (via $GET), but have not yet been started. The $BEGIN need not immediately follow the $GET. The effect of the $GET...$BEGIN sequence is the same as the $RUN command; the main purpose is to allow the user to insert changes into the program, which has been loaded, but not begun. The $MODIFY command is used to make these changes.

Example:

```
$BEGIN
```

Start executing a program at the normal start address.

CHANGE 2.1.4.3 $CHANGE

Format:

$CH[ANGE]{./,/Δ} dataset

Purpose:

This command changes the batch stream input to the dataset specified. This permits data, source programs, etc., to be stored on datasets other than the one used normally for batch input, and then to be read in during execution of a job.

$CHANGE command is a synchronous command. When the $CHANGE command is honored, the batch stream is read from the secondary dataset. When an end-of-file is sensed on the secondary dataset, command input is resumed from the primary dataset, at the point following the $CH command.

Example:

	A (PRIMARY DATASET)	B (SECONDARY DATASET)
1.	$JOB AAA[2ØØ,2ØØ]	$RUN FORTRN
2.	$RUN MACRO	#PROGA,LP:<PROGA
3.	#FILEA<FILEA	.
4.	$CHANGE B	.
5.	$RUN LINK	(EOF)
6.	#FILEA,LP:<FILEA,PROGA,FTNLIB/L/E	
7.	$FINISH	

In this example, when EOF is encountered on dataset B, command input is resumed at command 5 of dataset A ($RUN LINK).

NOTE

$JOB and $CHANGE are not legal in the secondary dataset. The job will be aborted if either is encountered.

2.1.4.4 $CPY **CPY**

Function: Copies input dataset(s) to an output dataset.

Format: $CPY input dataset1[,input dataset2,...,input datasetn] To output dataset

Input: At least one dataset must be specified; more than one may be specified. Two or more input datasets are concatenated into one output dataset, if the output is a specific file. Otherwise, they are separate. (Refer to Part 12 of this handbook, which discusses the file utility package, PIP.)

Output: Only one dataset must be specified.

Examples:

1. $CPY DT2:*.OBJ TO SY:

   All files with extension OBJ, residing on DECtape unit 2, are copied to the system device.

2. $CPY DK1:FIL.EXT[3,17] TO MT2:NUFIL.EX1

   The file, on RK11 disk unit 1, FIL.EXT (belonging to user 17 of group 3) is copied to magnetic tape unit 2. The name of the copy on magnetic tape is NUFIL.EX1.

DATE 2.1.4.5 $DATE

Format:

$DA[TE]

Purpose:

This command requests that the current date be included in the job log.

The date will be printed in the dd-mmm-yy format. When entered via the batch stream, the $DATE command may be used solely to place the date of the job's execution in the log. When the $JOB card is processed, the date and time are put in the log.

The user can enter a date value through the keyboard, while Batch mode is running. To do this, type

```
CTRL/C
.DATE dd-mmm-yy
```

putting the correct date value in place of

dd-mmm-yy.

DEL 2.1.4.6 $DEL

Function: Deletes specified datasets.

Format: $DEL dataset1[,dataset2,...,datasetn]

At least one dataset must be specified. If no device is specified, the system device is assumed. If a device is specified, it is assumed for following datasets that do not have device specifiers, until a device is specified.

Examples:

1. $DEL A

   A file named A is deleted from the system device.

2. $DEL DT1:FILA.FTN,DK1:FLE.MAC,FLA.FTN

   FILA,FTN is deleted from DECtape unit 1; FLE.MAC and FLA.FTN are deleted from RK11 disk unit 1.

3. $DEL *.MAC

   All files with extension MAC are deleted from the system device.

2.1.4.7 $DIR — DIR

Function: Obtains a directory listing.

Format: $DIR [[input dataset(s)] TO [output dataset]]

Input: One or more input datasets can be specified. If omitted, the directory obtained is that of the user who is currently logged in. The default device is the system device.

Output: The default device is the keyboard.

Examples:

1. $DIR

   The current user's system device directory is printed at the keyboard.

2. $DIR DF:

   The current user's RF11 disk directory is printed at the keyboard.

3. $DIR DT1:,DK:[3,5] TO LP:

   The current user's directory on DECtape unit 1, and user [3,5]'s directory on the RK11 disk, are printed at the line printer.

4. $DIR TO LP:

   The current user's directory, on the system device, is printed at the line printer.

5. $DIR *.OBJ TO LP:

   A directory listing of all files with extension OBJ that belong to the current user, and that reside on the system device, is printed on the line printer.

DUMP 2.1.4.8 $DUMP

Format:

$DU[MP]{Δ / ,}LP:[,[O][,start addr Ø [,end addr]]]

Purpose:

The $DUMP command causes an absolute copy of a specified core area to be written out of core to the line printer. If no arguments, other than device name, are supplied, values are assumed by default; e.g., "O", dump from core to the line printer starting at address Ø. If no end address is specified, the highest word in memory is the default value. A $DUMP command is valid at any time; if issued during program execution, operations are suspended for the time needed to complete the dump. A $DUMP can be entered through the keyboard while in Batch mode.

EOD 2.1.4.9 $EOD

Format:

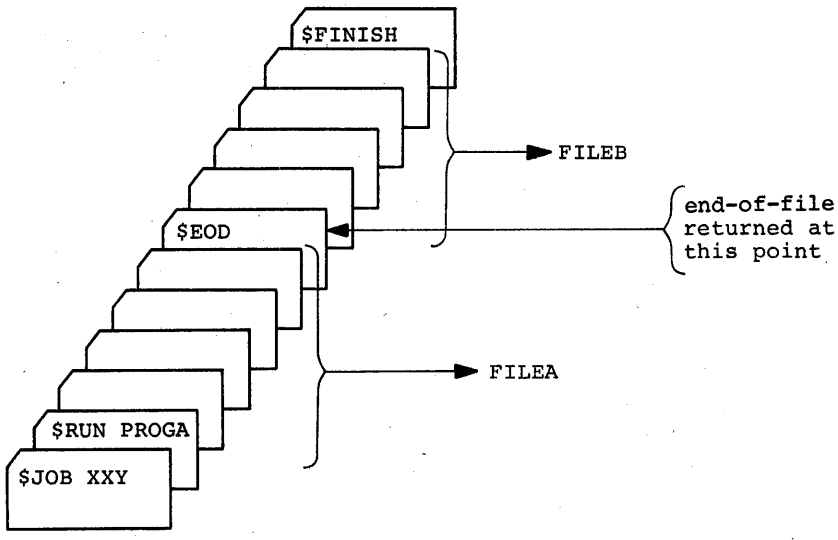
$EO[D]

Purpose:

The $EOD command stands for end-of-data. It delimits groups of data statements that are logically distinct, but physically contiguous.

When this statement is encountered in the batch stream, an end-of-file is generated and returned to the current stream. This end-of-file indicates that the data that follows is logically distinct from data the program has read to that point.

$EOD allows data to be stacked in a deck in physically contiguous fashion, while the program treats each group of cards as a logical unit. In the example below, the program PROGA processes data contained in two logically distinct datasets: FILEA, and FILEB. The $EOD card signals the end of FILEA, by returning end-of-file to PROGA when the last record in FILEA has been read. Then FILEB may be processed.



NOTE

$EOD provides a logical end-of-file; it is not the same as the physical end-of-file card, which is required to signal the end of a card file. (Refer to Appendix H.)

2.1.4.10 $EX[ECUTE]

EXECUTE

Function: Compiles a source program, links the object module, and runs the resulting load module.

Format: $EX[ECUTE] [source dataset] [TO [binary dataset] [,listing]]

Input: The source program is assumed to be FORTRAN. The FORTRAN Compiler is invoked to compile the source code specified in the input dataset.

If the source dataset is not specified, the source program is assumed to follow the $EXECUTE statement in the batch stream. An $EOD statement is required to signal the end of source input and the beginning of data.

Output: If the binary dataset and/or the listing dataset are omitted, the object and load modules are temporary files on the system device, and the listing is produced at the line printer. Otherwise, the object and load modules are produced as specified; i.e., the load module assumes the object module's filename, with extension .LDA.

Examples:

1. $EXECUTE

   The FORTRAN Compiler is loaded and run. The source program is read from the batch stream. The object module is output to the system device, linked into a load module, and run. The listing is printed at the line printer.

2. $EXECUTE PROG

   The FORTRAN Compiler is loaded and run, to compile the source program, PROG, from the system device. Linking and execution follow.

3. $EXECUTE DT1:ABC.CBA TO DK:ABC,LP:

   The FORTRAN Compiler is loaded and run to compile the source program ABC.CBA from DECtape unit 1. An object module (ABC.OBJ), and load module (ABC.LDA) are produced and placed on the RK11 disk. The load module is run. The listing is produced at the line printer as requested.

FINISH 2.1.4.11 $FINISH

Format:

$FI[NISH]

Purpose:

The $FINISH command delimits a job. When a $FINISH is detected, it signifies that the current job is ended. There are no arguments associated with $FINISH.

Processing continues with the next $JOB statement. Note that the $FINISH command cannot be entered through the keyboard while the system is in Batch mode.

To terminate the batch stream from the keyboard, type the following command.

```
CTRL/C
.TE
```

FORTRN 2.1.4.12 $FORTRN

Function: Loads and runs the FORTRAN Compiler to compile source input dataset (source) and produces a binary output dataset and listing.

Format: $FORTRN [[input dataset] TO [object dataset], [,list]]

Input: If omitted, the source program is assumed to follow immediately in the batch stream.

Output: The object dataset, if omitted, goes to the system device, with the file name specified in the input dataset. If no input file name was given, the object module assumes the job name. The extension is OBJ. The listing goes to the line printer, if defaulted.

Examples:

1. $FORTRN ABC TO XYZ,DT1:SRC

   The source program ABC is read from the system device and compiled, producing an object module. The object module is output to the system device, with the name XYZ,OBJ. The listing dataset goes to DECtape unit 1, with the name SRC.LST (LST is the default extension).

2. $FORTRN DK:ABC TO SY:ABC

   The source program ABC.FTN is read from the RK11 disk, and compiled; the object module goes to the system device, under the name ABC.OBJ; the listing is produced at the line printer by default.

3. $FORTRN ABC

   The source program ABC is read from the system device. The object module ABC.OBJ goes to the system device by default, and the listing is defaulted to the line printer.

4. $FORTRN

   The source program is read, immediately following in the batch stream (i.e., BI: is assumed). The binary object module is put on the system device with the listing at the line printer, both by default.

5. $FORTRN A TO ,LP:

   Generates a listing at the line printer, but no binary dataset.

2.1.4.13 $GET

GET

Format:

$$\$GE[T]\left\{\begin{matrix}\Delta\\,\end{matrix}\right\}\text{program specifier}\left[\ /DU\left[\left\{\begin{matrix}:\underline{PR}[OGRAM]\\:\overline{AL}[L]\\:V_1:V_2\end{matrix}\right\}\right]\right]$$

Purpose:

This command causes the program to be loaded into core from a specified device. The program specifier entry can include the device identifier, and the filename (and extension, if any) and uic of the program to be loaded. (A $BEGIN command would, at some point, normally follow the $GET, to start the program.) $GET can also be used in conjunction with the $SAVE command.

A dump-on-error switch (/DU) is available, allowing the user to obtain a dump of a specified area, in case an error occurs in his program. One of three values may be specified with the dump switch.

PR[OGRAM]	dump the program area
AL[L]	dump core in its entirety
$V_1:V_2$	dump the area delimited by the absolute octal addresses specified for V_1 and V_2. V_2 must be greater than V_1 and must be even (i.e., word boundaries).

The default value is PR. The dataset to which the dump is made is system-defined.

The program area depends on the location of the stack pointer.

1. If the stack pointer is below the load address, the program area is from the stack pointer to the top of core.
2. If the stack pointer is not below the load address, the program area is from the load address to the top of core.

The /DU switch forces an automatic dump in the event of an error, thus providing the Batch user a means of debugging his program. Refer to Chapter 4-3 for the format of error dumps.

JOB 2.1.4.14 $JOB

Function: Identifies the start of a Batch job, and permits the user to supply information pertinent to its execution.

Format:

$$\$JO[B] \left\{ \begin{matrix} \Delta \\ , \end{matrix} \right\} [jobname][uic]/sw1/sw2[,log\ dataset]$$

where

jobname — is specified by the user to assign a name to his job. This field consists of one or more letters or digits. The jobname could, for example, be the user's name (to help identify the destination of the log). Only the first six characters are used by the system. (The whole name is placed in the log, however.) If omitted, the job name is given a default value. (The first default job name is ØØØØØ1, the next is ØØØØØ2, etc.) Defaults are assigned in numerical sequence. Each time a batch session is started, the default sequence is reset.

[uic] is the field in which the user identifies himself by means of the user identification code. This field is delimited by left and right brackets. The format of the uic field is

[nnn,nnn]

where the nnn value to the left of the comma is an octal number identifying the user-group to which the user belongs; and the second nnn value, an octal number, identifies the particular user within that group. Thus, if the user has been assigned as user 27 within group 34, he would enter [34,27] for [uic].

If the uic is omitted, the default uic set by the system manager is used. If the default uic is Ø, the job is not run.

/sw1 is a switch used to set a limit on how long the job is permitted to run. It is formatted as

/TI:hh:mm

where hh and mm are specified as decimal integers for hours and minutes respectively. If no time limit is provided, the BATCH command value is assumed. A value greater than that specified in the BATCH command is ignored (see 4-1.3.1.X). If only one value is supplied, it is assumed to be minutes (see sample job card below).[1]

/sw2 is the switch that allows the user to suppress the log, a record of the job's execution. If /NL is specified for /sw2, no log (record) is produced of the job control statements processed during the job's execution. If /NL is omitted, the user will get this record as part of his output. The device used for the log is specified in the BATCH command (refer to 4-1.3.1).[1]

log dataset is an optional entry that specifies the dataset used for the log for this job. It overrides the log dataset specified in the BATCH command. /NL and a log dataset specification are mutually exclusive.

Example:

```
$JOB MATX[34,27]/TI:15,DT1:LOG
job name - MATX
user identification code - user group 34, user 27
time limit - 15 minutes
log - put the log on DECtape 1, under file name LOG.
```

[1]The switches may appear in either order on the $JOB card.

KILL 2.1.4.15 $KILL

Format:

```
$KI[LL]
```

Purpose:

The $KILL command terminates the current program, stops I/O and closes all open files. Processing continues at the next command prefixed by a $. No arguments are specified for $KILL.

The $KILL command can be entered via the keyboard to abort the job while in Batch mode; for example, to abort a job that is threatening to pre-empt system resources to the detriment of other jobs, type in the sequence shown here.

```
CTRL/C
.KILL
```

The following commands force a KILL.

```
$CHANGE
$FINISH
$GET
$JOB
$RUN
```

LINK 2.1.4.16 $LINK

Function: Invokes the LINK program, to link input datasets (object modules) into an output load module, and produce a load map.

Format: $LINK input dataset(s) TO load module [,load map]

Input: At least one input dataset is required; more than one can be specified. File characteristics, such as concatenated object modules (/CC) or library modules (/L), must be defined by the user (refer to Part 9, Linker).

Output: Complete dataset specification required; the load module extension is .LDA.

Example:

```
$LINK DF:A,DT1:FTNLIB/L TO SY:ABC,LP:
```

The object module A.OBJ is input from the RF11 disk, and linked with routines from FTNLIB.OBJ, input from DECtape unit. The load module ABC.LDA is output to the system device; the load map is produced at the line printer.

2.1.4.17 $LIST — LIST

Function: Prints datasets on the line printer.

Format: $LIST[dataset1,...,datasetn]

Input: If no datasets are specified, the dataset immediately following in the batch stream is printed on the line printer.

Output: Line printer only. Never specified.

Examples:

1. $LIST DK:A.FTN,B.FTN,DT1:Z.FTN

   Three datasets are printed at the line printer; DK:A.FTN, DK:B.FTN, and DT1:Z.FTN (DK: is used as the device specifier, until a different device is specified. If no device is specified, SY: is assumed.)

2. $LIST

   The dataset immediately following in the batch stream is printed.

2.1.4.18 $MACRO — MACRO

Function: Assembles MACRO source input, producing an object module and a listing as output.

Format: $MACRO input dataset(s) [TO [object dataset] [,listing dataset]]

Input: At least one input dataset is required; two or more can be specified. The MACRO Assembler assembles multiple input datasets together, creating a single object module.

NOTE

Input to the MACRO Assembler must be from a mass storage device. Source programs on punched cards must be copied to disk, DECtape, etc., prior to invoking the assembler.

Output: The object module defaults to the system device, and the listing defaults to the line printer.

Examples:

1. $MACRO A.MAC

   The source program A.MAC is assembled. The object module A.OBJ is output to the system device, by default. The listing is produced at the line printer, also by default.

2. $MACRO A TO DT1:Z,DK:A

   The source program A is assembled, to produce object module Z.OBJ, output to DECtape unit 1. The listing is placed on the RK11 disk, as A.LST.

3. $MACRO A.MAC TO ,LP:

The result of this form of $MACRO is a listing on the line printer. No object module is produced.

MESSAGE 2.1.4.19 $MESSAGE

Format:

$ME[SSAGE]

Purpose:

The $ME statement is used to send a message to the operator. A single message consists of one line. To send a message from the batch stream, the user inserts commands in the following format.

$ME (' / Δ) any text

The text can contain any ASCII characters the user wishes passed to the operator. The message is printed on the terminal.

Example of the $ME Usage:

Message lines in batch stream (job name TBLT);

```
$ME MOUNT DECTAPE LABELLED XYZ ON UNIT 1
$ME DO NOT WRITE ENABLE
$WAIT
```

Keyboard output:

```
TBLT : MOUNT DECTAPE LABELLED XYZ ON UNIT 1
TBLT : DO NOT WRITE ENABLE
AØ5Ø ØØØØØØ
$
```

The keyboard output includes the job name (TBLT) in the message line. The Action message (AØ5Ø ØØØØØØ) indicates that a $WAIT is in effect; the program is suspended until the operator types CO on the same line as the $.

The $WAIT statement is optional; if omitted, the message is printed at the keyboard, but the program continues without suspension. (Of course, the AØ5Ø ØØØØØØ, $ lines are not printed, since no operator response is expected.)

2.1.4.20 $MODIFY

MODIFY

Format:

$MO[DIFY]{ ' / ^ }octal addr:new contents

Purpose:

This command provides a way of changing the contents of an absolute memory location. Whatever was in the location is altered to the value specified in the new contents field. The system makes no provision for displaying the previous contents of the address. The value specified in the octal addr field must be an even number (i.e., aligned on a word boundary), and must not exceed 16 bits. Locations in the resident Monitor may not be modified.

Example:

$MO 2164Ø:16Ø4Ø

This changes the contents of location 2164Ø to 16Ø4Ø.

NOTE

The location specified must not fall within the area occupied by the Monitor or the job will be aborted.

2.1.4.21 $OWN

OWN

Format:

$OW[N]

Purpose:

This command causes the system to enter a mode of operation called OWN mode, which allows batch input data to be read in unformatted mode. Statements in the batch stream are treated as pure data; thus, special characters, such as $, #, and *, which might ordinarily cause some Monitor action to occur, are treated as data rather than control characters.

Currently, the $OWN command can be used only when the batch stream device is the card reader.

Return from OWN mode to normal input mode is effected by placing an end-of-file card at the end of the data.

NOTE

The characters $, #, and * can also be read as data, rather than as control characters, by placing an apostrophe in the first position of the line (refer to Section 4-2.4).

RESTART 2.1.4.22 $RESTART

Format:

$RE[START][(' address)]

Purpose:

This command permits a program to be restarted. As shown, the user may optionally supply an absolute address at which the program is to be restarted. Normally, a restart address will have been specified by the program. It is recommended that the address option for $RESTART be used with care.

$RESTART is valid only when the program is already loaded. Before the program is restarted, the stack is cleared, any current I/O is stopped, and all internal busy states are removed. Buffers and device drivers set up for I/O operations will, however, remain linked to the program for future use.

The $RESTART command is invalid if a restart address has not been specified, either by the program, or by an address field with the command itself. $RESTART may not be entered from the keyboard while the system is in Batch mode.

RNM 2.1.4.23 $RNM

Function: Renames an input file as specified in the output dataset.

Format: $RNM [device:]old name TO [device:]new name

Input, output: Both are required. They must both be on the same physical device. If omitted, the default is the system device.

Examples:

1. $RNM DT1:ABC TO DT1:XYZ

   The file ABC is renamed XYZ.

2. $RNM UNO TO DUE

   A file on the system device (UNO) is renamed DUE.

2.1.4.24 $RUN

RUN

Format:

$$\$RU[N]\ \left\{\begin{matrix}'\\ \Delta\end{matrix}\right\}\text{program specifier}\ \left[/DU\ \left\{\begin{matrix}:PR[OGRAM]\\ :AL[LL]\\ :V_1:V_2\end{matrix}\right\}\right]$$

Purpose:

The $RUN command causes a named program to be loaded from a specified device, and started at the normal address.

The program specifier provides the name of the program, and the device from which it is to be loaded, and, optionally, a user identification code that is associated with the program.

As with the $GET command, a dump-on-error switch may be included, in the format

$$/DU\ \left\{\begin{matrix}:PR[OGRAM]\\ :AL[L]\\ :V_1:V_2\end{matrix}\right\}$$

where

pr[ogram]	means dump the program area (see $GET).
AL[L]	means dump all of core.
$V_1:V_2$	means dump the area bounded by the absolute octal addresses specified for V_1 and V_2. V_1 and V_2 must be even values (i.e., word boundaries), and V_2 must be greater than V_1.

The default area is PROGRAM. If no /DU switch is included, no dump-on-error occurs. The dataset to which the dump is made is system defined.

Example:

```
$RU DT:PGM/DU
```

The program named PGM is loaded from DECtape, and started. The dump-on-error specification requests that the program area be dumped (by default).

2.1.4.25 $SAVE

SAVE

Format:

$$\$SA[VE]\left[,\text{dataset specifier}[/RA:low:high]\right]$$

Purpose:

The $SAVE command allows a program to be saved in loader format. It is used after a program has been loaded into core, prior to starting the program. The program is copied onto the device specified in the dataset specifier, under the name that is included in the dataset specifier, if any.

The $SAVE command may be used only if the program was never started. A common use is to load a program using $GET, insert fixes with $MODIFY, and then place the altered program onto secondary storage with a $SAVE command.

If no dataset specifier is included, the current program will be saved on the system disk, under the name SAVE.LDA. Any file previously saved under this name will first be deleted.

The /RA switch (range) is included so the user can save an absolute area other than that occupied by his current program. If he wishes to save only the current program area, this switch is omitted. Including the /RA switch saves only the specified area. The absolute addresses specified for /RA must be valid octal word boundary addresses.

The command will be rejected if an additional 256-word buffer cannot be allocated from free core.

Example:

```
$SA,REG.LDA
```

The $SAVE command in this example causes the current program to be saved on the system disk, under the name REG.LDA.

TIME 2.1.4.26 $TIME

Format:

```
$TI[ME]
```

Purpose:

Including the $TIME command provides a means of obtaining the time-of-day in the output job log. It does not permit the user to specify a time from the batch stream. Attempting to do so is illegal. (The current job will be aborted.)

To enter a time value, the user should type the following at the keyboard.

```
CTRL/C
$TIME hh:mm:ss
```

This is a valid entry from the keyboard while the system is operating in Batch mode.

The time-of-day is also placed in the job log by the $FINISH and $JOB commands. The $JOB command also includes the current date.

2.1.4.27 $WAIT **WAIT**

Format:

```
$WA[IT]
```

Purpose:

This command suspends processing, and causes the Action message

```
AØ5Ø ØØØØØØ
$
```

to be printed at the keyboard. It is usually used in conjunction with the $ME command. To resume operation, type CO.

2.2 INPUT TO COMMAND STRING INTERPRETER

The Command String Interpreter (CSI) accepts command strings consisting of dataset specifications. The purpose of a command string is to establish the datasets to be used for input and output by a particular program. In Figure 4-3 (see Chapter 4-1) the third card (#MATX,LP:<BI:) is a command string. As indicated there, the first character must be #, in column 1 (or in line position 2, if input is not from a card). The format of a CSI command is

```
#output dataset(s)<input dataset(s)
```

where a dataset can be specified as follows.

$$dev:filename.ext\ [uic]\ /sw_1:V_1:\ldots:V_n/sw_n:V_1:\ldots:V_n$$

Each dataset specification is delimited by a comma. The elements comprising a dataset specification provide information concerning the dataset's location, its filename and extension (if it is a file), the user identification code associated with the file, and any switches that may be used to specify particular actions to be performed. Device specifiers are selected from those listed in Table 4-2 (refer to Chapter 4-1). The ability to include the pseudo-device specifiers (BI and SY) is a feature of Batch that provides device-independence when specifying datasets to the Command String Interpreter. It allows a dataset to be specified without requiring that the user know what device may actually be used at job execution time.

For example, a user may have his source data on cards, but because of the greater speed to be gained by reading the data from a faster device, he may transcribe the data onto another storage medium, such as disk. He would then specify the disk to be the batch stream input device.

```
$RUN FORTRAN          run the FORTRAN Compiler;

#DK:PRG,LP:<BI:       specify that the input dataset is located
                      in the batch stream.
```

The same command string can be used whether the batch stream is coming from disk, magnetic tape, DECtape, paper tape or cards. By specifying BI, the user has ensured that the command string is valid for all these devices; there is no need to change the card to match the specific device.

The SY device specifier is used to designate the system residence device, as in the following example.

```
#SY:FILE.FTN<PR:
```

In this example, a command string to PIP specifies that a dataset is to be input from the paper-tape reader, and output to the system-residence device. This command string is valid, whatever the system-residence device may be when PIP is executed.

2.3 SYSTEM PROGRAM COMMANDS

Commands that are directed to system programs are identified by an asterisk in position 1. For example, to issue the Insert command to EDIT, the user must include a command in the format

```
*I
```

followed by the text to be inserted. Refer to the appropriate manual for details on the commands used in system programs.

2.4 READING CONTROL CHARACTERS AS DATA

The characters $, #, and *, appearing in the first position of a line (or card column 1), are interpreted as control characters, and are stripped off before the remainder of the line is passed to its destination (Monitor, Command String Interpreter, or system program). It may happen that the user wishes to include one of these characters as actual data, to be passed along with the rest of the data on the line, rather than having it stripped off. To do this, place an apostrophe in the first position,

```
'$AMT 1ØØ
```

which causes the line to be passed as $AMT 1ØØ (the apostrophe is stripped). This may also be accomplished through $OWN (see 4-2.1.4.21).

If the apostrophe is not found in column 1, but the $ is there instead, the card would be treated as a command to the Monitor.

Valid control statements can be included, to cause the Monitor, CSI, or system program to take a desired action. Thus, a deck of cards being read by a user program may include a statement such as

```
$RUN PIP
```

to invoke the Peripheral Interchange Program. When the $RUN PIP card is encountered, an EOF is returned, and the card is held until a READ is issued to the command input dataset (CMI); at this point, the $RUN PIP card is passed to the Monitor, which causes the user program to be terminated, and PIP to be loaded and executed. (Refer to the discussion of synchronous/asynchronous commands, in Section 4-2.1.3.)

PART 4

CHAPTER 3

INPUT/OUTPUT

3.1 BATCH INPUT

Batch input can be from any PDP-11 device that can perform the input function. Data can be read from the batch stream in one of two ways: normal mode or OWN mode.

3.1.1 Normal Input Mode

In most cases the normal input mode is employed when reading data from the batch stream. Data read in normal mode must be formatted data. Attempting to read unformatted data, if not in OWN mode, results in a fatal error, aborting the job. (See the DOS/BATCH Monitor, Part 3 of this handbook, for a definition of formatted data.)

This requirement stems from the need to check the first position on each line for the presence of a control character. Because it is impossible to determine the beginning and end of a line in unformatted data, a situation could arise where the control card could be inadvertently bypassed, causing unpredictable results. Note: All formatted reads from the batch stream must have a byte count of at least 83 specified in the line buffer header maximum byte count word.

3.1.2 OWN Mode

It is occasionally necessary to read in unformatted mode from the batch stream; e.g., when translating EBCDIC characters to ASCII. This is permitted by means of the $OWN control card, which indicates to the Monitor that all characters read from that point until the next physical end-of-file (EOF) card (which terminates OWN mode) may be read as unformatted data. In this mode, the characters $, #, and * are not treated as control characters, but as data.

The physical end-of-file (EOF) card statement must be included at the end of the user's OWN data to avoid the possibility of failing to recognize a control card. The operator must place an EOF card at the end of each job to prevent the next job from being read as data, in case the prior user forgot to terminate OWN mode. For added safety, the user should place an EOF card immediately ahead of his $JOB card. Binary data within a batch stream from the card reader must be read in OWN mode.

3.2 BATCH OUTPUT

Batch output includes program listings, output associated with system programs (such as load maps), the job log, and dumps.

3.2.1 Job Log

The job log is the record of events that occurred during execution of the job: the control cards processed, commands read, and error messages generated. The first line of the job log contains the image of the $JOB command, as specified by the user, the date, and the time. This is followed by a sequence of images of control cards that were read in and processed up to the point at which the $FINISH command was read, or a fatal error occurred. Any error, warning, or informative messages are included in the log as they are encountered. Provided that a log dataset was included in the BATCH command, a log is produced for all jobs, unless the log-suppress switch (/NL) was specified in the $JOB card.

3.2.2 Dumps

If the user has specified the /DU switch on the $RUN or $GET card for the program, and an error occurs, a dump of the area specified in the /DU switch is produced. The first page of a dump, the header page, consists of a summary of information regarding the dump itself. It is formatted as shown in Figure 4-4.

```
          * * * * * * DUMP OF HEADER FOLLOWS * * * * * *

STARTING WORD OF DUMP = nnnnnn
NO. OF BYTES DUMPED = nnnnnn
R0 = nnnnnn
R1 = nnnnnn
R2 = nnnnnn
R3 = nnnnnn
R4 = nnnnnn
R5 = nnnnnn
SP = nnnnnn
PC = nnnnnn
HIGH ADDRESS = nnnnnn
LOW ADDRESS = nnnnnn

DUMP IDENTIFIER MESSAGE
```

Figure 4-4
Sample Header Page

The values of nnnnnn are given in octal, and are left-justified. For example, the value 477 in the NO. OF BYTES DUMPED entry would appear as NO. OF BYTES DUMPED = 477.

The subsequent pages of the dump comprise the area specified in the /DU switch. As shown in Figure 4-5, each group of four lines describes 1ØØ(8) locations, and is headed by the flag --- n ---, where n corresponds to the first location in the group, Ø, 1ØØ, 2ØØ, etc.

Each line in the dump contains the contents of eight words, represented as octal value. If a sequence of lines contains all zeroes, the first line in the sequence is printed in its entirety.

The next line is printed with asterisks in the first position.

```
4Ø:  ******
```

Subsequent lines in the sequence are omitted altogether. The next nonzero line is printed.

At the end of each line is a field of 16 ASCII characters, which is the ASCII contents of each byte. ASCII equivalents of characters that are not within the printable range are mapped into the printable range, converted to upper case (if necessary), and printed.

Figure 4-5 illustrates the form and content of a dump.

```
* * * * * * DUMP OF DATA FOLLOWS * * * * * *


---000000---
000: 010214 000024 006102 000342 006102 000344 006102 000346: LPT•BLB•BLD•BLF•
020: 006116 000340 006136 000340 000400 000340 006102 000352: NL••AL•••A••BLJ•
040: 006216 001020 002072 002124 002204 002312 002354 002526: HLPBZDTDDDJDLDVE
060: 002726 000200 002726 000201 006102 000356 006102 000356: VE••VEA•BLN•BLN•


---000100---
000: 007262 000300 006102 000356 006102 000356 153170 000356: RN••BLN•BLN•XVN•
020: 006102 000356 006102 000356 006102 000356 006102 000356: BLN•BLN•BLN•BLN•
040: 006102 000356 006102 000356 006102 000356 006102 000356: BLN•BLN•BLN•BLN•
060: 006102 000356 006102 000356 006102 000356 006102 000356: BLN•BLN•BLN•BLN•


---000200---
000: 011506 000200 006102 000356 006102 000356 006102 000356: FS••BLN•BLN•BLN•
020: 007024 000240 006102 000356 006102 000356 006102 000356: IN••BLN•BLN•BLN•
040: 006102 000356 006102 000356 006102 000356 006102 000356: BLN•BLN•BLN•BLN•
060: 006102 000356 006102 000356 006102 000356 006102 000356: BLN•BLN•BLN•BLN•


---000300---
000: 006102 000356 006102 000356 006102 000356 006102 000356: BLN•BLN•BLN•BLN•
020: 006102 000356 006102 000356 006102 000356 006102 000356: BLN•BLN•BLN•BLN•
040: 006102 000356 006102 000356 006102 000356 006102 000356: BLN•BLN•BLN•BLN•
060: 006102 000356 006102 000356 006102 000356 006102 000356: BLN•BLN•BLN•BLN•


---000400---
000: 004567 001466 005005 010601 062701 000014 012102 012146: WIVCEJAQAEL•BTFT
020: 042716 177437 011645 105715 100001 111715 014204 012703: NE••ESMKA•MSDXCU
040: 001402 120467 005645 101033 005000 120467 005034 101014: BCWAEK[B•JWA•KLB
060: 005743 013100 001421 152715 000340 005710 001023 130504: CK•VWCMU••HKSBDQ


---000500---
000: 001424 010210 111015 022341 006304 062704 176332 011404: ICHPMSADULDEZ•DS
020: 010416 000004 103432 012607 120427 000006 001764 010216: NQDLZGGUWAF•TCNP
040: 010346 000004 105404 011102 000401 005722 012625 005741: FPD•DKBRAARKUUAK
060: 010100 106204 100707 103401 022520 014110 010240 010541: •PDLGAAGPEHX•PAQ


---000600---
000: 004567 001320 062606 000002 011646 030516 001750 012704: WIPBFEB•FSNQHCDU
020: 001060 012403 121227 000030 103405 121227 000040 103002: PBCUWBX•EGWB••BF
040: 012704 003302 151715 012343 001046 021624 001012 005714: DUBFMSCTFBTCJBLK
060: 100002 105714 001040 005224 014116 041716 020021 112615: B•LK•BTJNXNCQ•MU
```

Figure 4- 5
Sample Data Dump

PART 4

CHAPTER 4

BATCH PROGRAMMING CONVENTIONS

This chapter is aimed at programmers responsible for writing, modifying, or maintaining programs that function in the Batch mode operating system environment. Familiarity with DOS/BATCH is assumed.

To function properly in Batch mode, programs must observe certain conventions. Currently existing programs should be modified to conform to these conventions, if the user intends them to operate in the Batch environment.

As an aid in making the modifications, assemble the program to be modified with MACRO and obtain a CREF listing. The cross-reference data will help in locating and examining relevant link blocks and file blocks.

4.1 COMMAND DATASETS

The Monitor organizes BATCH input into several unique pseudo datasets based on the identifying character at the beginning of each record, i.e.:

CHARACTER	PSEUDO DATASET
$	BOS
#	CMI
*	PCI
any other	CDI

All BATCH output is handled through the CMO pseudo dataset. These pseudo datasets have been implemented to ensure the operating system has constant control over the input to each program. This control is maintained through a hierarchal command structure, i.e., a CMI input dataset is not allowed if a BOS dataset has not been input previously, or if a BOS dataset is received after a PCI dataset then the PCI dataset is discontinued and the BOS dataset regains control.

4.1.1 Command String Input (CMI)

The dataset named CMI is used for all Command String Interpreter input; i.e., all commands with a # symbol in line position (or card column) 1. When reading CMI, a .WAIT should follow the .READ, and a test for end-of-data (EOD) should be made; if EOD has occurred, the Monitor EXIT EMT should be issued.

4.1.2 Command Output (CMO)

The dataset with logical name CMO is used for:

a. All Command String Interpreter related output (such as syntax error announcements, and the # symbol);

b. all responses to program command input (e.g., *);

c. all error logging.

A default must be specified for the CMO physical device. This default must be

KB:

so that the program will run in either DOS or BATCH mode.

NOTE

Because the physical devices for the datasets used for command string input and output will, in all likelihood, not be the same; it is recommended that a .WRITE to CMO be followed by a .WAIT, before issuing a .READ to CMI, and vice-versa.

4.1.3 Program Command Input (PCI)

All program command input must be entered via a dataset named PCI. That is, all commands prefixed by * must be read from PCI. When end-of-data (EOD) is detected from PCI, the proper procedure is to clean up the current Command String Interpreter request (# command) and read the next # command.

4.1.4 Command Data Input (CDI)

Input to programs, other than program commands, such as text insert to EDIT, must be entered via a link block that has a logical dataset named CDI. When EOD on CDI is detected, the current * command processing has finished. The next * command should be read from PCI.

4.2 READS FROM BATCH STREAM

As described in Chapter 3, all READS from the batch stream must be formatted, unless the $OWN command has been issued. All formatted READS from the batch stream must have a byte count in the line buffer header of $\geq 83_{10}$. This requirement precludes corruption of commands that may be read by the user. (If the byte count is less than 83_{10}, a command on a card might be truncated before being read in its entirety.)

4.3 PSEUDO DEVICE SPECIFIERS

The device specifiers BI and SY are used when the user wishes to call for the Batch input device, or the system residence device, but he does not know which device is actually being used. Since these specifiers do not call for a physical device, in the way that CR or DF do, they are termed pseudo device specifiers. They allow the system to supply the actual device that is being used at job execution time, in place of BI or SY. Thus, the same control cards can be used, regardless of the particular device being employed.

4.4 USE OF ASSIGN

The ASSIGN statement must observe the rules listed below.

1. $ASSIGN must not be used with any Batch system program's logical dataset name. The user must not assign a dataset to CMI, CMO, DCI, or PCI.

2. An $ASSIGN that is made at the job level is global to the job. The sequence

```
$JOB MAC [200,200]
$ASSIGN DT0:CRT,RDO
```

   causes a file named CRT, which is on DECtape, to be assigned to the dataset with logical name RDO. This assignment, if not altered by a later ASSIGN in the job, remains in effect for the duration of the job (MAC).

3. An $ASSIGN that is made at the program level remains in effect (if not subsequently altered) for the duration of the program.

```
$GET PROG
$ASSIGN DK:MTX.OBJ,DCL
$BEGIN
```

   The assignment of file MTX.OBJ, on the RK11 disk, to dataset DCL is in effect for the duration of PROG.

4.5 NOTE PERTAINING TO .CSI2 RETURN CONDITIONS

The user should note that on return from .CSI2, the top of the stack may have bit 2 set. Bit 2 is set when a default device is returned by the Command String Interpreter; i.e., the user has not specified a device in a command string, but has chosen to use the default device, instead.

As documented in Part 3, DOS/BATCH Monitor, the user is only required to check bits Ø and 1. In cases where this is done by checking but 1 (to determine that no error occurred) and then checking the value of the word for a zero or nonzero value, the presence of a 1 in bit 2 may lead to erroneous assumptions.

4.6 ERROR HANDLING

The command output dataset (CMO) must be used for output of all error announcements that come directly from a system program, rather than via an IOT. Direct error announcements include announcements of command string syntax errors, and supplementary information (such as filenames) concerning error announcements made through an IOT.

An EMT has been incorporated into the Batch system to allow the currently running program (system or user) to request that lines in the batch stream be bypassed, until a specified type of control card is encountered. For example, if a command string syntax occurs, it may be desired to bypass all following statements up to the next $ or # statement. This EMT is incorporated by including the following:

```
MOV #CODE,-(SP)
EMT 67
```

where CODE's value determines the next statement type to be read (not bypassed).

CODE	STATEMENT TYPE
Ø	$
1	$ or #
2	$, #, or *

Any other value for CODE is invalid, and causes a fatal error (FØ53).

The DOS/BATCH convention of announcing syntax errors by printing the command as far as the point of the error, followed by ?, is still used in Batch, but the EMT shown above must be included to cause the batch stream to be bypassed from the point of the error until a line starting with $, or # is found. (CODE = 1) EMT 67 acts as a NOP when the system is not in Batch mode.

PART 4

CHAPTER 5

BATCH CARDS

5.1 CARD CODES

Each card input to the DOS/BATCH system must contain either an ASCII, Ø26, or Ø29 punch code which represents either a single command to the Monitor or a single line of data. (See Appendix A for a list of the card codes and their corresponding meaning.)

The default card code is from the Ø29 keypunch. This may be changed by the user during the dialogue at system initialization. (See the System Manager's Guide DEC-11-OSMGA-A-D for more information.) The default code may be over-ridden at run time with coded header cards in the input deck. These header cards contain a special multipunch in column 1. A deck punched in Ø26 code should be preceded by a card containing the multipunch 12-2-4-8; and Ø29 deck requires the multipunch 12-Ø-2-4-6-8, and an ASCII deck uses the 12-1-3-5-8-9 in column 1. A deck can contain the various card codes (i.e., ASCII, Ø26 (BCD), Ø29 (EDCDIC) if a header card precedes the corresponding section of cards.

Table 5-1
HEADER CARDS

Multipunch	Function
12-2-4-8	Indicates that the cards which follow are to be read as Ø26 punch codes.
12-Ø-2-4-6-8	Indicates that the following cards are punched in Ø29 code.
12-1-3-5-8-9	Indicates that the following cards contain ASCII punches.
12-11-1-2-8-9	Enables blank suppression.
12-11-Ø-1-6-7-8-9	Indicates end of file and must conclude each input file

5.2 READ LIMITATIONS

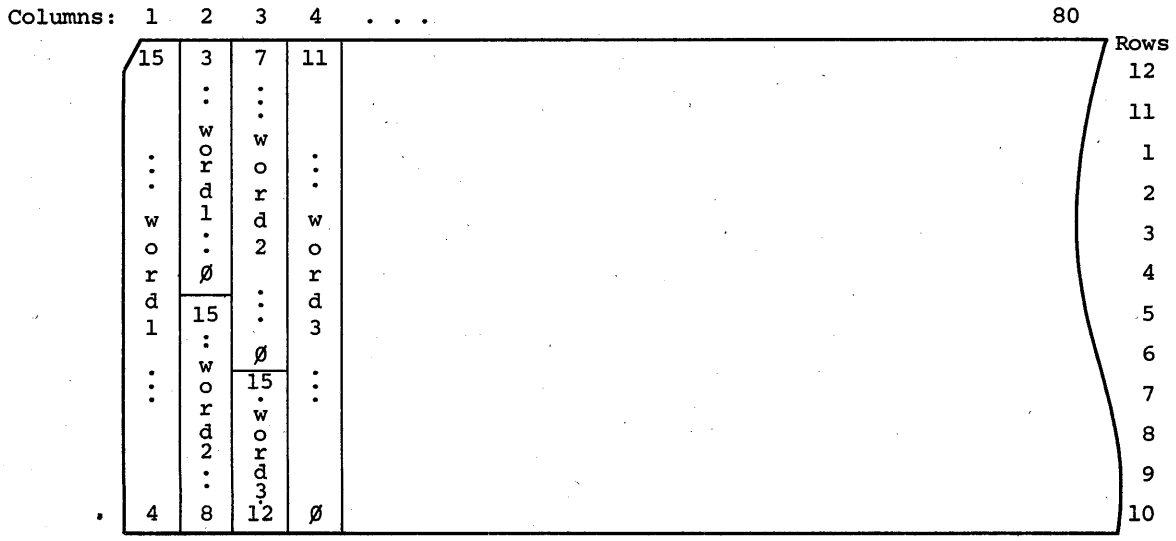
For each I/O request, eighty ASCII characters are stored in a specified user buffer. The Monitor supplies a carriage return and line feed at the end of the buffer. If the data being input exceeds the length of the buffer, excess data is discarded and a carriage return and line feed character are placed at the end of the full buffer.

One method of over-riding the 80-character ASCII read limit is with blank suppress. Blank suppression is enabled by a header card which contains the multipunch 12-11-1-2-8-9 in column 1. With blank suppress enabled, columns 73-80 on the input cards are ignored. The carriage return and line feed which terminate the input are set immediately after the last non-blank data column. This permits the user to eliminate trailing blanks and create more compact records.

5.3 BINARY DATA

The user can input binary data by specifying the transfer mode in the Mode Byte of the line buffer header for .READ or by setting Bit Ø of the Function/Status word to 1 in the Tran Block for .TRAN level I/O requests (see Monitor Chapter 3-4 for more information). Programs which use .READ in unformatted binary mode or .TRAN must make their own checks for data validity. Monitor processing checks data validity in formatted binary .READ requests if the data conforms to normal standards.

Binary input data is accepted as 12 bits per column and passed to the calling program in packed form as four columns which build three binary words as illustrated below.



Each punch represents one bit of a binary word, however, a word is 16 bits and a card column has only 12 rows, consequently bits are packed in descending order for each word. Bits 15 through 4 of the first word fall into the first column. The remaining 4 bits (i.e., 3 to Ø) and the high order bits of the second word fall into the second column. This process continues allowing 60 words of binary data to be packed into a single 80 column card.

5.4 ERROR CONDITIONS

The detection of any card reader error condition in Batch signifies a "Device not Ready" state which elicits a AØØ2 message and disables the reader interrupt. If the operator issues a CONTINUE command to resume processing, the error processor will recall the Transfer routine to repeat the read and exit to await the next interrupt. The operator is given the opportunity to correct the error before entering the CONTINUE command. The card with the error should be replaced and the replacement card should be the first card read when processing resumes. An exception to this procedure occurs whenever the A37Ø diagnostic message is printed. In this case, the last card from the output side should not be replaced in the input hopper for it has already been read.

A "Hopper Empty" condition indicates a "Device not ready" state as opposed to an end of medium condition. The "Hopper Empty" condition signifies that more data can be input; this allows input decks to be larger than the size of the Hopper. This condition necessitates the use of a header card to indicate the end of file. The end of file card contains the multipunch 12-11-0-1-6-7-8-9 in the first column for ASCII mode transfer or in each of the first eight columns for binary mode.

As a "Hopper Empty" condition is detected before the last card has been processed, it is essential that an EOF card be followed by one or more blank cards. If the blank cards are initially omitted, normal completion can only be effected by reinserting the EOF card followed by a blank card.