DEC-10-OMPBA-A-D

decsystem10

BEGINNER'S GUIDE TO MULTIPROGRAM BATCH

digital equipment corporation · maynard. massachusetts

1st Edition May 1972

FOREWORD

The Beginner's Guide to Multiprogram Batch has been written for the inexperienced or casual user who has little knowledge of programming techniques and who requires only a rudimentary knowledge of Batch operations.

HOW TO USE THIS MANUAL

For those users whose mode of input is cards, the following chapters or sections of chapters should be read.

Chapter 1	Introduction
Chapter 2	Entering a Job to Batch from Cards
Chapter 4	Interpreting Your Printed Output
Chapter 5, Section 5.2	Using Cards to Enter Jobs

According to the language in which his program is written, the user should pay particular attention to the following sections.

FORTRAN	-	Section 2.2.3 Card Deck to Run FORTRAN Programs
ALGOL	-	Section 2.2.1 Card Deck to Run ALGOL Programs
COBOL	-	Section 2.2.2 Card Deck to Run COBOL Programs
MACRO	-	Section 2.2.4 Card Deck to Run MACRO Programs
BASIC	-	Section 2.3.1 Card Decks for Programs That Do Not Have Special Control Cards

For users who input their jobs through interactive terminals, the following chapters or sections of chapters are recommended.

Chapter 1	Introduction
Chapter 3	Entering a Job to Batch from a Terminal
Chapter 4	Interpreting Your Printed Output
Chapter 5, Section 5.1	Using the Terminal to Enter Jobs

REFERENCES

Not all of the commands and cards for Batch are described in this manual. Those users who wish to know more about Multiprogram Batch can refer to Chapter 3 in the DECsystem-10 Operating System Commands manual. Also in that manual, the SUBMIT command is described in Chapter 2.

An elementary description of the basic monitor commands can be found in the document Getting Started with Timesharing. The DECsystem-10 Operating System Commands manual contains the descriptions of all the monitor commands available to the user.

Error messages from the system programs supplied by DEC that are invoked by the user's job are explained in the applicable manuals. For example, if a user's FORTRAN program fails to compile successfully, the error messages he receives from the FORTRAN compiler can be found in Chapter 11 of the FORTRAN IV Programmer's Reference Manual in the DECsystem-10 Mathematical Languages Handbook.

CONVENTIONS USED IN THIS MANUAL

The following is a list of symbols and conventions used in this manual.

dd-mmm-yy hhmm	A set of numbers or numbers and a word that indicates the date and time, e.g., 15-5-72 1415 or 15-MAY-72 1415 means 2:15 PM on May 15, 1972.
filename.ext	The name and extension that can be put on a file. The name can be 1 to 6 characters in length and the extension can be 1 to 4 characters in length. The first character of the extension must always be a period. The extension is optional. Refer to the glossary for definitions of filename and filename extension.
hh:mm:ss	A set of numbers representing time in the form hours:minutes:seconds. Leading zeros can be omitted, but colons must be present between two numbers. For example, 5:35:20 means five hours, 35 minutes, and 20 seconds.
jobname	The name that is assigned to a job. It can contain up to 6 characters. Refer to the glossary for the definition of a job.
[proj,prog]	The user number assigned to each user, commonly called a project-programmer number. It must be enclosed in square brackets. The two numbers that make up the project-programmer number must be separated by a comma or a slash. Refer to the glossary for the definition of a project-programmer number.
n	A number that specifies either a required number or an amount of things such as cards or line-printer pages. This number can contain as many digits as are necessary to specify the amount required, e.g., 5, 25, 125, etc.
t	A number representing an amount of time usually in minutes. This number can contain as many digits as are necessary to specify the amount of time required, e.g., 5, 25, 125, etc.

GLOSSARY

Term	Definition
ALGOL	ALGOrithmic Language. A scientific oriented language that contains a complete syntax for describing computational algorithms.
Alphanumeric	The characters which include the letter of the alphabet (A through Z), the numerals (0 through 9), and letters of the other special symbols such as -, /, *, $, ., (,), +.
ASCII Code	American Standard Code for Information Interchange. A 7-bit code in which information is recorded.
Assemble	To prepare a machine-language program from a symbolic-language program by substituting absolute operation codes for symbolic operation codes and absolute or relocatable addresses for symbolic addresses.
Assembler	A program which accepts symbolic code and translates it into machine instruction, item by item. The assembler on the DECsystem-10 is called the MACRO assembler.
Assembly Language	The machine-oriented symbolic programming language belonging to an assembly system. The assembly language for the DEC-system-10 is MACRO.
Assembly Listing	A printed list which is the byproduct of an assembly run. It lists in logical-instruction sequence all details of a routine showing the coded and symbolic notation next to the actual assigned notations established by the assembly procedure.
BASIC	Beginner's All-purpose Symbolic Instruction Code. A time-sharing computer programming language that is used for direct communication between teletype units and remotely located computer centers. The language is similar to FORTRAN II and was developed by Dartmouth College.
Batch processing	The technique of executing a set of computer programs in an unattended mode.
Card	A punch card with 80 vertical columns representing 80 characters. Each column is divided into two sections one with character positions labeled zero through nine, and the other labeled eleven (11) and twelve (12). The 11 and 12 positions are also referred to as the X and Y zone punches, respectively.

GLOSSARY (Cont)

Term	Definition
Card Column	One of the vertical lines of punching positions on a punched card.
Card Field	A fixed number of consecutive card columns assigned to a unit of information.
Card Row	One of the horizontal lines of punching positions on a punched card.
Central processing unit (CPU)	The portion of the computer that contains the arithmetic, logical, control circuits, and I/O interface of the basic system.
Central Site	The location of the central computer. Used in conjunction with remote communications to mean the location of the DECsystem-10 central processor.
Character	One symbol of a set of elementary symbols such as those corresponding to the keys on a typewriter. The symbols usually include the decimal digits 0 through 9, the letters A through Z, punctutation marks, operation symbols, and any other special symbols which a computer may read, store, or write.
COBOL	COmmon Business Oriented Language. An automatic programming language used in programming data processing applications.
Command	The part of an instruction that causes the computer to execute a specified operation.
Compile	To produce a machine or intermediate language routine from a routine written in a high level source language.
Compiler	A programming system which translates a high level source language into a language suitable for a particular machine. A compiler is a translator that converts a source language program into intermediate or machine language. Some compilers used on the DECsystem-10 are: ALGOL, BASIC, COBOL, FORTRAN.
Computer	A device with self-contained memory capable of accepting information, processing the information, and outputting results.
Computer Operator	A person who manipulates the controls of a computer and performs all operational functions that are required in a computing system, such as, loading a tape transport, placing cards in the input hopper, removing printouts from the printer rack, and so forth.
Continuation Card	A punched card which contains information that was started on a previous punched card.
Control File	The file made by the user that directs Batch in the processing of his job.

GLOSSARY (Cont)

Term	Definition
Core Storage	A storage device normally used for main memory in a computer.
CPU	See central processing unit.
Cross Reference Listing	A printed listing that identifies all references of an assembled program to a specific label. This listing is provided immediately after a source program has been assembled.
Data	A general term used to denote any or all facts, numbers, letters, and symbols, or facts that refer to or describe an object, idea, condition, situation, or other factors. It represents basic elements of information which can be processed or produced by a computer.
Debug	To locate and correct any mistakes in a computer program.
Disk	A form of mass storage device in which information is stored in named files.
Dump	A listing of all variables and their values, or a listing of the values of all locations in core.
Execute	To interpret an instruction and perform the indicated operation(s).
Extension	See filename extension.
File	An ordered collection of 36-bit words comprising computer instructions and/or data. A file can be of any length, limited only by the available space on the device and the user's maximum space allotment on that device.
Filename	A name of one to six alphanumeric characters chosen by the user to identify a file.
Filename extension	One to four alphanumeric characters usually chosen to describe the class of information in a file. The first character of the extension must always be a period.
FORTRAN	FORmula TRANslator. A procedure oriented programming language that was designed for solving scientific type problems. The language is widely used in many areas of engineering, mathematics, physics, chemistry, biology, psychology, industry, military, and business.
Job	The entire sequence of steps, from beginning to end, that the user initiates from his interactive terminal or card deck or that the operator initiates from his operator's console.

GLOSSARY (Cont)

Term	Definition
Jobstep	A serial or parallel sequence of processes invoked by a user to perform an operation.
K	A symbol used to represent a thousand; for example, 32K is equivalent to 32,000.
Label	A symbolic name used to identify a statement in the control file.
Log File	A file into which Batch writes a record of a user's entire job. This file is printed as the final step in Batch's processing of a job.
Monitor	The collection of programs which schedules and controls the operation of user and system programs.
Monitor Command	An instruction to the monitor to perform an operation.
Mounting a device	To request assignment of an I/O device via the operator.
Multiprogramming	A technique that allows scheduling in such a way that more than one job is in an executable state at any one time.
Object Program	The program which is the output of compilation or assembly. Often the object program is a machine language program ready for execution.
Password	The word assigned to a user that, along with his user number (project-programmer number), identifies him uniquely to the system.
Peripheral devices	Any unit of equipment, distinct from the central processing unit, which can provide the system with outside communication.
Project-programmer number	Two numbers separated by a comma, which, when considered as a unit, identify the user and his file storage area.
Program	The complete plan for the solution of a problem, more specifically the complete sequence of machine instructions and routines necessary to solve a problem.
Programming	The science of translating a problem from its physical environment to a language that a computer can understand and obey. The process of planning the procedure for solving a problem. This may involve among other things the analysis of the problem, preparation of a flowchart, coding of the problem, establishing input-output formats, establishing testing and checkout procedures, allocation of storage, preparation of documentation, and supervision of the running of the program on a computer.

GLOSSARY (Cont)

Term	Definition
Queue	A list of jobs to be scheduled or run according to system, operator, or user-assigned priorities. For example, the Batch input queue.
Software	The totality of programs and routines used to expend the capabilities of computers, such as compilers, assemblers, operational programs, service routines, utility routines, and subroutines.
Source Deck	A card deck comprising a computer program, in symbolic language.
Source Language	The original form in which a program is prepared prior to processing by the computer.
Source Program	A computer program written in a language designed for ease of expression of a class of problems or procedures, by humans. A translator (assembler, compiler, or interpreter) is used to perform the mechanics of translating the source program into a machine language program that can be run on a computer.
Terminal	A keyboard unit that is often used to enter information into a computer and to accept output from a computer. It is often used as a time-sharing terminal on a remotely located computer center.

CONTENTS

Page

CHAPTER 1 INTRODUCTION
1.1 What is Multiprogram Batch 99
1.2 How to Use Batch 99
1.2.1 Setting Up Your Job 99
1.2.2 Running Your Job 100
1.2.3 Receiving Your Output 100
1.2.4 Recovering from Errors 100
1.3 Summary 100

CHAPTER 2 ENTERING A JOB TO BATCH FROM CARDS
2.1 Format of the Cards in Your Deck 103
2.2 Setting up Your Card Deck 104
2.2.1 Card Deck to Run ALGOL Programs 105
2.2.2 Card Deck to Run COBOL Programs 106
2.2.3 Card Deck to Run FORTRAN Programs 106
2.2.4 Card Deck to Run MACRO Programs 107
2.3 Putting Commands into the Control File from Cards 108
2.3.1 Card Decks for Programs that do not have Special Control Cards 109
2.4 Control Cards for Batch (in Alphabetical Order) 111
2.4.1 The $ALGOL Card
2.4.2 The $COBOL Card 113
2.4.3 The $DATA Card 115
2.4.4 The $DECK Card 119
2.4.5 The End-of-File Card 121
2.4.6 The $EOD Card 121
2.4.7 The $ERROR Card 122
2.4.8 The $FORTRAN Card 123
2.4.9 The $JOB Card 125
2.4.10 The $MACRO Card 127
2.4.11 The $NOERROR Card 129
2.4.12 The $PASSWORD Card 130
2.4.13 The $SEQUENCE Card 131
2.5 Specifying Error Recovery in the Control File 131

CHAPTER 3 ENTERING A JOB TO BATCH FROM A TERMINAL
3.1 Creating the Control File 137
3.1.1 Format of Lines in the Control File 138
3.2 Submitting the Job to Batch 139
3.2.1 Queue Operation Switches 140
3.2.2 General Switches 141

CONTENTS (Cont)

		Page
3.2.3	File-Control Switches	143
3.2.4	Examples of Submitting Jobs	144
3.3	Batch Commands (in Alphabetic Order)	146
3.3.1	The .BACKTO Command	146
3.3.2	The .ERROR Command	146
3.3.3	The .GOTO Command	147
3.3.4	The .IF Command	148
3.3.5	The .NOERROR Command	149
3.4	Specifying Error Recovery in the Control File	150
CHAPTER 4	INTERPRETING YOUR PRINTED OUTPUT	
4.1	Output from Your Job	153
4.2	Batch Output	153
4.3	Other Printed Output	154
4.4	Sample Batch Output	154
4.4.1	Sample Output from a Job on Cards	154
4.4.2	Sample Output from a Job from a Terminal	157
4.4.3	Sample Dump	160
CHAPTER 5	PERFORMING COMMON TASKS WITH BATCH	169
5.1	Using the Terminal to Enter Jobs	169
5.2	Using Cards to Enter Jobs	176

CHAPTER 1
INTRODUCTION

1.1 WHAT IS MULTIPROGRAM BATCH

Multiprogram Batch is a group of programs that allow you to submit a job to the DECsystem-10 on a leave-it basis. That is, you give the job to an operator (if on cards) or submit it directly to the computer (if from a timesharing terminal) so that you can do something else while your job is running. A job is any combination of programs, their associated data, and commands necessary to control the programs.

Some of the jobs that are commonly processed under Batch are those that:

1. Are frequently run for production,
2. Are large and long running,
3. Require large amounts of data, or
4. Need no actions by you when they are running.

1.2 HOW TO USE BATCH

Batch allows you to submit your job to the computer through either an operator or a timesharing terminal, and receive your output from the operator when the job has finished. Output is never returned at a timesharing terminal even if your job is entered from one; instead, it is sent to a peripheral device (normally the line printer) at the computer site and returned to you in the manner designated by the installation manager.

1.2.1 Setting Up Your Job

You must make up a control file to use Batch. A control file is a list of commands for the monitor, system programs, or Batch itself that tells Batch what steps to follow to process your job and the order in which to process them. When you enter your job on cards, you can take advantage of the special control cards that cause Batch to insert commands into the control file for you. When you enter your job from a timesharing terminal, you must put all the commands for your job into the control file yourself. The steps that you must take to create a control file from cards are described in Chapter 2. Creating a control file from a timesharing terminal is described in Chapter 3.

1.2.2 Running Your Job

After you submit the job, it waits in a queue with other jobs until Batch schedules it to run under guidelines established by the installation manager. Some factors that affect how long your job waits in the queue are its size, the amount of core it needs, the amount of time that it will take to run it, and whether or not you have specified a certain deadline when you want it run. When the job is started, Batch reads the control file and performs the actions necessary to run the job. For example, Batch passes monitor commands to the monitor which performs the actions called for and passes commands to system programs so that their processing can be performed.

As each step in the control file is performed, Batch records it in a log file. For example, if a monitor command such as COMPILE is processed, Batch passes it to the monitor and writes it in the log file. The monitor response is also written in the log file. Any response from your job that would be written on the terminal during timesharing is written in the log file by Batch.

1.2.3 Receiving Your Output

When the job is completed successfully and output has been sent to all devices, Batch terminates the log file and has it printed. The output from your job and the log file are then returned to you. Output from your job can be in the form of line-printer listings, punched cards, punched paper tape, plots, DECtape, or magnetic tape. If the output is to a DECtape or magnetic tape, you must include commands in your job to mount these tapes so that your output can be written on them. This is also true if you have input to any of the programs in your job written on tape. If your output is to cards, paper tape, the plotter, or the line printer, you must specify to Batch the approximate amount of cards, paper tape, plotter time, or pages that you require. These restrictions are to help Batch restrain runaway programs. An example of using the MOUNT command in the control file to request mounting of tapes is shown in Chapter 5. The way that you specify the amounts of paper, cards, etc. is described in Chapter 2, "The $JOB Card" and in Chapter 3, "Submitting Your Job."

1.2.4 Recovering from Errors

If an error occurs in your job, either from an error in your program or from an erroneous command in the control file, Batch writes the error message in the log file and usually terminates the job. In addition, if the error occurred in your program, Batch causes a dump to be taken of your area of core. You can, however, put commands in the control file so that Batch can help you recover from errors in your job and continue running. Error recovery from a card job is described in Chapter 2; from a job entered from a terminal, in Chapter 3. Dumps, along with other printed output from a Batch job, are described in Chapter 4.

1.3 SUMMARY

In summary, the steps that you must perform to enter a job to the computer through Batch are as follows:

1. Create a control file either from cards (refer to Chapter 2) or from a terminal (refer to Chapter 3).
2. Submit the job to Batch, either to the operator for a card job (Chapter 2) or directly to Batch for a terminal (Chapter 3).
3. Pick up your output and interpret it (refer to Chapter 4).

Some sample jobs that are run through Batch from cards and from a terminal are shown in Chapter 5.

CHAPTER 2
ENTERING A JOB TO BATCH FROM CARDS

Batch runs your job by reading a control file that contains commands to the monitor, system programs, or Batch itself. You have to make up the control file, but Batch provides you with special control cards to help you make up control files for simple jobs. These control cards make it easy for you to submit your programs to the computer and to create your control file to run these programs. Most of these control cards cause Batch to insert commands into the control file and/or copy programs and data into disk files. Some are used to show the beginning of your job and to identify it; and one is used to indicate the end of it. Batch control cards are also available to help you recover from errors that may occur while you job is running. The following is a list of the control cards which are described in greater detail in Section 2.4.

$SEQUENCE	Section 2.4.12
$JOB	Section 2.4.9
$PASSWORD	Section 2.4.12
$ALGOL	Section 2.4.1
$COBOL	Section 2.4.2
$FORTRAN	Section 2.4.8
$MACRO	Section 2.4.10
$DECK	Section 2.4.4
$DATA	Section 2.4.3
$EOD	Section 2.4.6
$ERROR	Section 2.4.7
$NOERROR	Section 2.4.11
end-of-file	Section 2.4.5

2.1 FORMAT OF THE CARDS IN YOUR DECK

The card decks that you input to Batch can contain any combination of Batch control cards; commands to the monitor, system programs, and Batch itself; programs and data that will be copied into separate disk files; and data that will be copied into the control file for your program to read.

The Batch control cards must contain a dollar sign ($) in column 1 and a command that starts in column 2. The command must be followed by at least one space, which can then be followed by the other information on the card. The end-of-file is the only exception to this format; it is identified by special punches in columns 1 and 80. Refer to the individual description of each card for any special format requirements.

If you include a card with a monitor command, you must place a period in column 1 and follow it immediately with the command. Any information that follows the command is in the format that is shown for the command in the DECsystem-10 Operating System Commands manual.

To include a command to a system program on a card, you must punch an asterisk (*) in column 1 and punch the command string immediately following the asterisk. Refer to the manual for the system program that you wish to use.

Batch commands are punched like monitor commands; that is, a period is punched in column 1 and the command immediately follows the period.

The card format for your program depends on the language in which you have written the program; refer to the reference manual for the programming language that you are using for the format of each line of your program. The same is true for your data. The format that is required for the data by the programming language that you are using is described in the language reference manual.

If you want to include data for your program in the control file, you punch it as you would data that is read from a separate file. The only restriction on data in the control file is that alphabetic data that is preceded by a dollar sign ($) must be preceded by an additional dollar sign so that Batch will not interpret such data as its own control commands.

If you put any special characters other than those described above in the first column of a card, you may get unexpected results because Batch interprets other special characters in special ways. If you want to know about other special characters, refer to the DECsystem-10 Operation System Commands manual, Chapter 3.

If you have more information than will fit on one card, you can continue on the next card by placing a hyphen (-) as the last nonspace character on the card to be continued and the rest of the information on the next card.

Comments can also be included either as separate cards or on cards containing other information. To include a comment on a separate card, you must punch a semicolon (;) in column 1 and follow it immediately with the comment. To add a comment to a card, you must precede the comment with a semicolon (;) after all the information that you need has been put on the card. Comments that are on separate cards will normally be copied by Batch into your control file and later copied into your log file.

2.2 SETTING UP YOUR CARD DECK

Since the most common tasks performed in a job are compilation and execution of one or more programs, simple control cards are available that will cause Batch to insert commands into the control file for these tasks. However, a Batch job can do anything a timesharing job can do and if you wish to perform more complicated tasks, you will have to include monitor commands in your deck to direct

Batch to execute your tasks. The way in which you include monitor commands and also commands to other system programs is described in Section 2.3.

The control cards that you can use to compile and execute programs written in ALGOL, COBOL, FORTRAN, and MACRO are shown in sections 2.2.1, 2.2.2, 2.2.3, and 2.2.4. Certain control cards are always required in a Batch job. Others are required only at some installations. The $JOB card and the end-of-file card are always required. The $SEQUENCE and $PASSWORD cards may be required, depending on the installation.

If the $SEQUENCE card is required, it must be the first card in the deck. The $JOB card must always be either the second card in the deck if the $SEQUENCE card is required, or the first card in the deck if the $SEQUENCE card is not required. If it is required, the $PASSWORD card must immediately follow the $JOB card. It will be assumed in this manual that the $SEQUENCE and the $PASSWORD cards are required. The end-of-file card must be the last card in the deck to indicate to Batch that it has read the end of your job. This end-of-file card is only used to end your entire job, not to end individual files in your job.

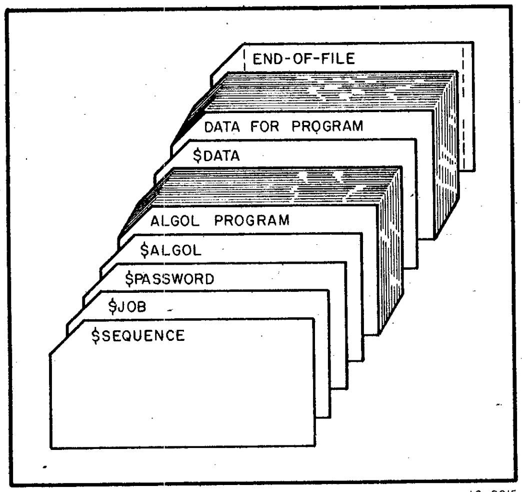
The cards that come between the first and last cards constitute your job. Setting up decks for specific languages is shown in the sections that follow.

2.2.1 Card Deck to Run ALGOL Programs

To run ALGOL programs, you use the $ALGOL and $DATA cards. You put a $ALGOL card in front of your ALGOL program to make Batch copy your program into a disk file and insert a COMPILE command into your control file. The $ALGOL card is described in detail in Section 2.4.1.

You put a $DATA card in front of the data that goes with the program to make Batch copy your data into another disk file and insert an EXECUTE command into your control file. The $DATA card is described in Section 2.4.3.

Thus, to compile and execute an ALGOL program, your card deck would appear as follows.



10-0915

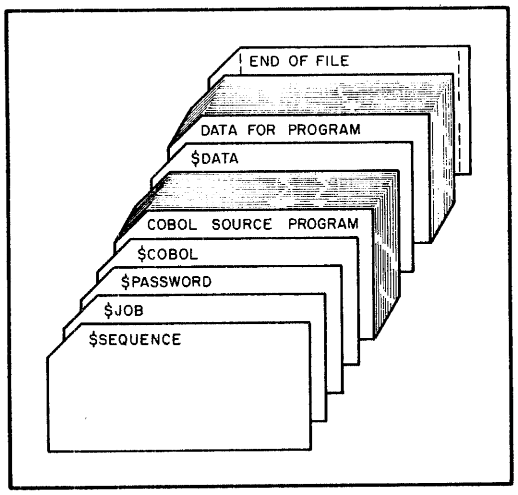
Refer to the description of each card for the information that goes on it. The way that you tell your program how to find its data is described in Section 2.4.3.1.

2.2.2 Card Deck to Run COBOL Programs

To run COBOL programs, you can use the $COBOL card and the $DATA card. You put a $COBOL card in front of your COBOL program to make Batch copy your program into a disk file and insert a COMPILE command into your control file. The $COBOL card is described in detail in Section 2.4.2.

You put a $DATA card in front of the data that goes with your program to make Batch copy your data into another disk file and insert an EXECUTE command into your control file. The $DATA card is described in Section 2.4.3.

Thus, to compile and execute one COBOL program, your card deck would appear as follows.



10-0916

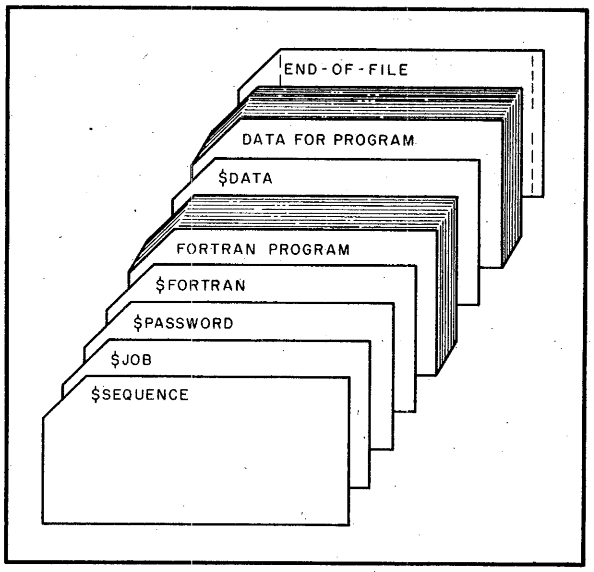
Refer to the description of each card for the information that goes on it. The way that you tell your program how to find its data is described in Section 2.4.3.1

2.2.3 Card Deck to Run FORTRAN Programs

To run FORTRAN programs, you can use the $FORTRAN and $DATA cards. You put a $FORTRAN card in front of your FORTRAN program to make Batch copy your program into a disk file and insert a COMPILE command into your control file. The $FORTRAN card is described in detail in Section 2.4.8.

You put a $DATA card in front of the data that goes with your program to make Batch copy your data into another disk file and insert an EXECUTE command into your control file. The $DATA card is described in Section 2.4.3.

Thus, to compile and execute one FORTRAN program, your card deck would appear as follows.



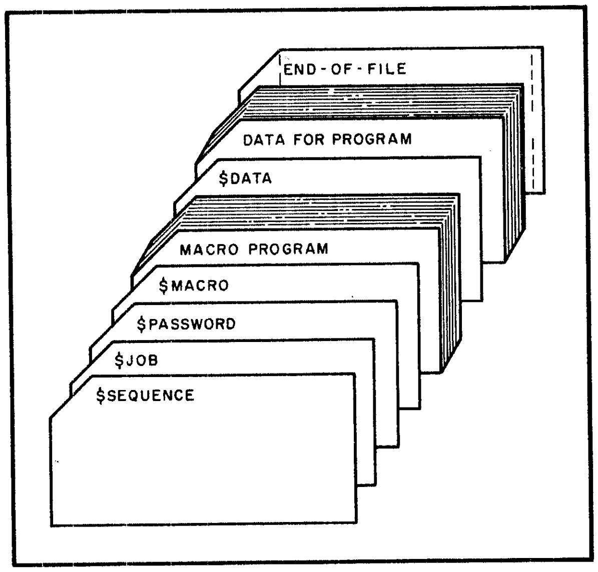
10-0917

Refer to the description of each card for the information that goes on it. The way that you tell your program how to find its data is described in Section 2.4.3.1.

2.2.4 Card Deck to Run MACRO Programs

To run MACRO programs, you can use the $MACRO and $DATA cards. You put a $MACRO card in front of your MACRO program to make Batch copy your program into a disk file and insert a COMPILE command into your control file. The $MACRO card is described in detail in Section 2.4.10.

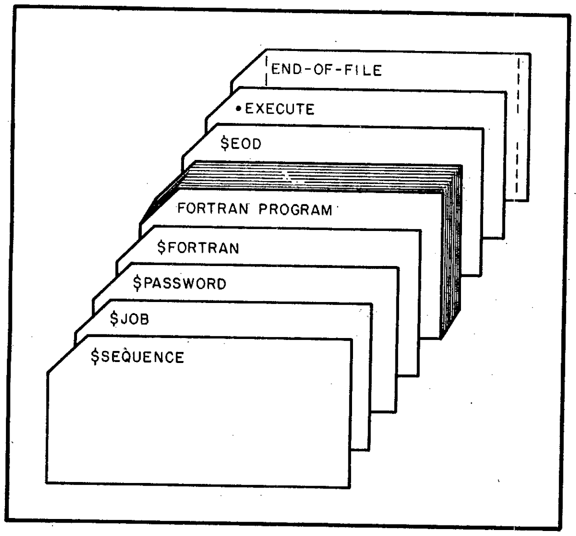
You put a $DATA card in front of the data that goes with your program to make Batch copy your data into another disk file and insert an EXECUTE command into your control file. The $DATA card is described in Section 2.4.3. Thus, to assemble and execute one MACRO program, your card deck would appear as follows.



Refer to the description of each card for the information that goes on it.

2.3 PUTTING COMMANDS INTO THE CONTROL FILE FROM CARDS

Batch puts commands into the control file for you when you use certain control cards. However, only a small number of kinds of commands can be put in the control file in this way. If you wish to perform operations in addition to compilation and execution, you must include commands in your card deck so that Batch will copy them into your control file. Where you put these commands in your card deck determines their positions in the control file. Batch reads your card deck in sequential order, copying commands into the control file as they, or the special control cards, are read. However, Batch, when it reads a control card that tells it to copy a program or data into a disk file, copies every card that follows such a control card until it meets another control card. To ensure that your commands are not copied into a file with programs or data, you must place a special control card, the $EOD card, at the end of a program deck if you wish to follow the program with a command. For example, if you have a FORTRAN program that creates its own data and does not need to use a $DATA card, you could include the following cards in your deck.



10-0919

(command to load and execute the program)
(to tell Batch to stop copying into the program file)

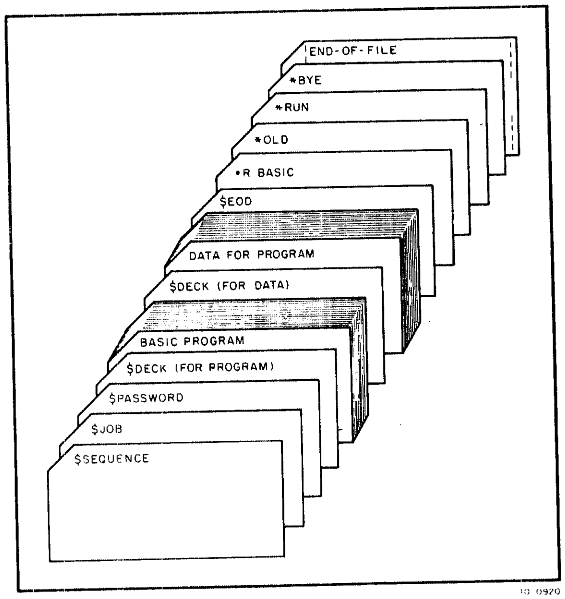
The only commands that you cannot use in a Batch job are CSTART, CCONT, ATTACH, DETACH, and SEND. Batch will ignore these commands when it reads them in the control file. Also, you cannot use the LOGIN command in your Batch job because you will get an error that will terminate your job. Batch logs your job in according to your $JOB and $PASSWORD cards.

2.3.1 Card Decks for Programs That Do Not Have Special Control Cards

By combining monitor commands with control cards such as $DECK and $EOD, in addition to the required control cards, you can process any program that does not have special control cards for it. You put a $DECK card in front of a program, data, or any other group of cards to make Batch copy the cards that follow the $DECK card into a disk file. However, Batch does not put a command into the control file when it reads a $DECK card. The $DECK card is described in detail in Section 2.4.4.

For example, a BASIC program does not have a specific control card. To run a BASIC program under Batch from cards, you can combine the $DECK card and the $EOD card with monitor commands. You also use a $DECK card to copy the data for a BASIC program because the $DATA card puts an EXECUTE command into the control file and BASIC does not use the EXECUTE command to run.

The following example shows a card deck that enters a BASIC program for running under Batch.

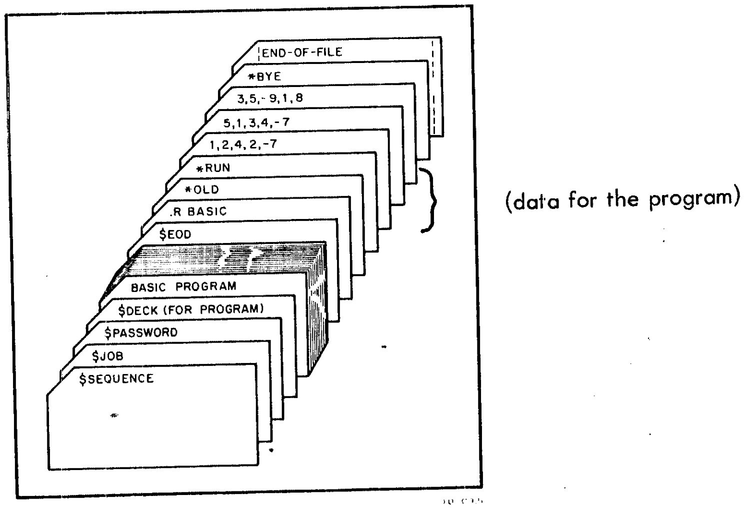


The BASIC program contains statements that read data from a disk file. You answer OLD to the BASIC question

NEW OR OLD --

because the file is on disk and can be retrieved by BASIC.

If your BASIC program reads data that is to be input by you during the running of the program, you enter the data in the control file so that it will be passed to your program by Batch. This is shown in the following example.



You can use the same technique to enter programs written in any language that does not have a specific control card, provided that your installation supports the language. Also, you can run system programs under Batch using the same technique.

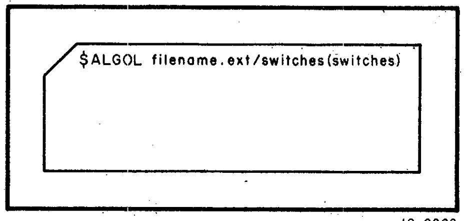
2.4 CONTROL CARDS FOR BATCH (IN ALPHABETICAL ORDER)

The special control cards for Batch are described below in detail. Only the control cards that are pertinent to this manual are discussed. Refer to DECsystem-10 Operating System Command (DEC-10-MRDC-D) for the remaining cards. The same is true for some of the switches that can be included on each card. If a switch is not described in this manual, it can be found in the DECsystem-10 Operating System Commands manual.

2.4.1 The $ALGOL Card

You put a $ALGOL card in front of your ALGOL program to make Batch copy your ALGOL program into a disk file and insert a COMPILE command into your control file. Thus, when Batch runs your job, your ALGOL program will be compiled. You can put some optional information on the $ALGOL card to tell Batch more about your program or the cards that your program is punched on.

The $ALGOL card has the form:



10-0902

filename.ext	specifies the optional filename and extension that you can tell Batch to put on the file that it creates for your program. If you omit the filename and extension, Batch will create a unique name for your file and add the extension .ALG to it.
/switches	are switches to Batch to tell it how to read your program and whether or not to request a compilation listing when the program is compiled. The switches can be put on the card in any order and are described below.
(switches)	are switches that Batch passes to the ALGOL compiler when it puts the COMPILE command in the control file. These switches must be enclosed in parentheses, must not be preceded by slashes, and may or may not be separated by commas. The switches for the ALGOL compiler are described in Section 18.1 in Chapter 18 of the DECsystem-10 ALGOL Programmer's Reference Manual (DEC-10-KAZB-D).

/WIDTH:n Switch

Normally, Batch reads up to 80 columns on every card of the ALGOL program. You can make Batch stop reading at a specific column by means of the /WIDTH switch, in which you indicate the number of a column at which to stop. Thus, if you have no useful information in the last 10 columns of each card of your program, you can tell Batch to read only up to column 70 by specifying

```
/WIDTH:70
```

on the $ALGOL card.

/NOLIST Switch

Normally, the $ALGOL card tells Batch to ask the compiler to generate a compilation listing of your ALGOL program. The listing is then printed as part of your job's output. If you don't want this listing, you can include the /NOLIST switch on the ALGOL card to stop generation of the listing.

/SUPPRESS:OFF Switch

When Batch reads the cards of your ALGOL program, it normally does not copy any trailing spaces into the disk file to save space on the disk. If you want Batch to copy everything on the cards up to column 80 or any column that you may specify in the /WIDTH switch, you must include the /SUPPRESS:OFF switch on the $ALGOL card.

Examples

The simplest form of the $ALGOL card is shown in the following example.

```
$ALGOL
```

This card causes Batch to copy your program into a file to which Batch gives a unique name and the extension .ALG. The cards in the program are read up to column 80 with trailing spaces suppressed. A listing file is produced when the program is compiled. This listing is written as part of the job's output. No compiler switches are passed to ALGOL.

The following is an example of a $ALGOL card with switches.

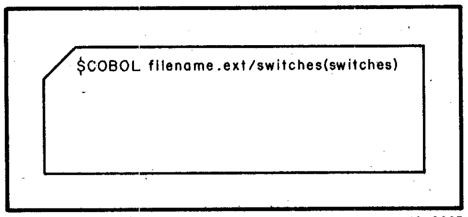
```
$ALGOL MYPROG.ALG /WIDTH:72 /NOLIST (1000D,N,Q)
```

With this card, your ALGOL program is copied into a file named MYPROG.ALG and a COMPILE command is entered into the control file. The cards in the program are read up to column 72 and trailing spaces up to column 72 are not copied into the file. When the program is compiled, no listing is produced, and the compiler reads and acts upon the switches 1000D,N,and Q given to it by Batch.

2.4.2 The $COBOL Card

You place the $COBOL card in front of your COBOL program to make Batch copy your COBOL program into a disk file and insert a COMPILE command into your control file. Thus, when Batch runs your job, your COBOL program will be compiled. You can put some optional information on the $COBOL card to tell Batch more about your program or the cards that your program is punched on.

The $COBOL card has the form:



```
$COBOL filename.ext/switches(switches)
```

10-0903

filename.ext	specifies the optional filename and extension that you can tell Batch to put on the file that it creates for your program. If you omit the filename and extension, Batch will create a unique name for your file and add the extension .CBL to it.
/switches	are switches to Batch to tell it how to read your program. The switches are described below.
(switches)	are switches that Batch passes to the COBOL compiler when it puts the COMPILE command in the control file. These switches must be enclosed in parentheses, must not be preceded by slashes, and may or may not be separated by commas. The switches for the COBOL compiler are described in Table D-3 in Appendix D of the DECsystem-10 COBOL Programmer's Reference Manual (DEC-10-KC1C-D).

/WIDTH:n Switch

Normally, Batch reads up to 80 columns on every card of the COBOL program. You can make Batch stop reading at a specific column by means of the /WIDTH switch, in which you indicate the number of a column at which to stop. Thus, if you have no useful information in the last 10 columns of each card of your program, you can tell Batch to read only up to column 70 by specifying

```
/WIDTH:70
```

on the $COBOL card.

/SUPPRESS:OFF Switch

When Batch reads the cards of your COBOL program, it normally does not copy any trailing spaces into the disk file to save space on the disk. If you want Batch to copy everything on the card up to column 80 or any column that you may specify in the /WIDTH switch, you must include the /SUPPRESS:OFF switch on the $COBOL card.

/CREF Switch

If you want a cross-reference listing of your COBOL program, you can include the /CREF switch on the $COBOL card to tell Batch to ask the COBOL compiler to produce a cross-reference listing when it compiles your program. This listing is printed as part of your job's output. You do not have to include a command to run the CREF program to get this listing, Batch will do it for you.

/SEQUENCE Switch

The COBOL compiler assumes that your COBOL program is in standard DECsystem-10 format. The /SEQUENCE switch, which Batch passes to the compiler, makes the compiler recognize that your program is in conventional (i.e., industry-wide) format. A program in conventional format has sequence numbers in columns 1 through 6 and comments that begin in column 73. When the /SEQUENCE switch is specified, the width of the card is assumed to be 72, not 80 columns. The following example shows programs in conventional and standard formats.

IF YOUR PROGRAM LOOKS LIKE:	YOU SHOULD:

```
1      8                          73
000010 IDENTIFICATION DIVISION..... MYPROG
000020 PROGRAM-ID. MYPROG........ MYPROG
000030 AUTHOR. ABB.................. MYPROG
   .                                 .
   .                                 .
   .                                 .
```
	Include the /SEQUENCE switch because your program is in conventional format.

```
1
IDENTIFICATION DIVISION......
PROGRAM-ID. MYPROG.........
AUTHOR. ABB..................
.
.
.
```
	Omit the /SEQUENCE switch because your program is in DECsystem-10 standard format.

Examples

The simplest form of the $COBOL card is:

```
$COBOL
```

This card tells Batch to copy your program into a file and assign a unique name and the extension .CBL. All 80 columns of the cards are read, trailing spaces are not copied, and the compiler is told that the program is in standard format. No switches are passed to the compiler, and a listing file is produced when the job is run. The listing is printed as part of the job's output.

The following is an example of a $COBOL card with switches.

```
$COBOL MYPROG.CBL /SEQUENCE (N,P)
```

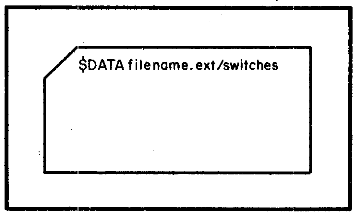
With this card, your COBOL program is copied into a disk file named MYPROG.CBL and a COMPILE command is inserted into the control file. The cards are read only up to column 72 and trailing spaces up to column 72 are not copied into your file. Batch passes the N and P switches to the compiler, and tells the compiler to accept the program in conventional format. A listing file is produced when the program is compiled. This listing is printed as part of the job's output.

2.4.3 The $DATA Card

You put a $DATA card in front of the data for your program to make Batch copy it into a disk file and to insert an EXECUTE command into your control file. Within the EXECUTE command, Batch requests a loader map for you. When your job is run, any programs that were entered with $ALGOL, $FORTRAN, or $MACRO cards that came before the $DATA card are executed. Every time that Batch reads one of the $language cards, it adds it to a list that it keeps. When it then reads a $DATA card, each program in Batch's list is put into the EXECUTE command string that the $DATA card puts into the control file. After the $DATA card is read by Batch and the EXECUTE command is put into the control file with the names of the programs that preceded the $DATA card, Batch clears its list so that it can start a new list for programs entered later. If you have more than one set of data for a program or programs, you can precede each set with a $DATA card to put two EXECUTE commands into the control file to run your program or programs twice. An EXECUTE command following another EXECUTE command in the control file without intervening $language cards causes the programs executed by the first EXECUTE command to be loaded and executed again.

If your data is included in the program so that you do not have cards with data on them, you can still use the $DATA card to insert an EXECUTE command into the control file.

The form of the $DATA card is:



10-0904

filename.ext	specifies the optional filename and extension that you can tell Batch to put on the file that it creates for your data. If you omit the filename and extension, Batch will create a unique name for your file and add the extension .CDR to it.

/switches	are switches to Batch to tell it how to read the cards of your data. The switches are described below.

/WIDTH:n Switch

Normally, Batch reads up to 80 columns on every card of your data. You can make Batch stop reading at a specific column by means of the /WIDTH switch, in which you indicate the number of a column at which to stop. Thus, if you have information in the last 10 columns of each card of your data, you can tell Batch to read only up to column 70 by specifying

```
        /WIDTH:70
```

on the $DATA card.

/SUPPRESS:OFF Switch

When Batch reads the cards of your data, it normally does not copy any trailing spaces into the disk file to save space on the disk. If you want Batch to copy everything on the cards up to column 80 or any column that you specify in the /WIDTH switch, you must include the /SUPPRESS:OFF switch on the $DATA card.

Examples

The simplest form of the $DATA card is:

```
        $DATA
```

This card causes Batch to copy your data into a file and to assign a unique name and the extension .CDR to it. All 80 columns of the cards are read and trailing spaces are not copied into the file.

The following example shows a $DATA card with switches.

```
        $DATA MYDAT.DAT /WIDTH:72 SUPPRESS:OFF
```

The data that follows this card is copied into a file named MYDAT.DAT and an EXECUTE command is inserted into the control file. When Batch reads the cards of the data, it reads only up to column 72 and copies trailing spaces into the data file.

2.4.3.1 Naming Data Files on the $DATA Card - If you want to name your data file on the $DATA card rather than letting Batch name it for you, you must, in your program, assign that file to disk as shown in the following examples.

COBOL Example

```
               :
ENVIRONMENT DIVISION.
INPUT-OUTPUT SECTION.
SELECT SALES, ASSIGN TO DSK.
               :
DATA DIVISION.
FILE SECTION.
FD SALES, VALUE OF IDENTIFICATION IS "SALES CDS".
               :
```

The $DATA card would then appear as follows.

```
$DATA SALES.CDS
```

FORTRAN Examples

You can assign your data to disk in several ways when you use FORTRAN. You can read from unit 1, which is the disk, in your program and use the name FOR01.DAT as the filename on your $DATA card, as shown in the following statements.

```
          .
          .
          .
READ (1,f), list
          .
          .
          .
$DATA FOR01.DAT
```

You can also tell FORTRAN to read from logical unit 2, which is normally the card reader, and assign unit 2 or the card reader (CDR) to disk (DSK). You use the name FOR02.DAT on the $DATA card in this case.

```
          .
          .
          .
READ (2,f), list
          .
          .
          .
.ASSIGN DSK CDR (in the control file)
$DATA FOR02.DAT
```

You can also use a specific disk device such as DSK0 as the unit from which you will read. In the control file, you would then assign DSK0 to DSK. The unit number of DSK0 is 20 and thus the name on the $DATA card would be FOR20.DAT.

```
        .
        .
        .
READ (20,f), list
        .
        .
        .
.ASSIGN DSK DSKO (in the control file)
$DATA FOR20.DAT
```

ALGOL Example

To read your data from the disk in an ALGOL program, you would use the following statements. You can assign your data to any channel (signified by c) and you can give your data file any name as long as the name that you use in your program is the same as that put on the $DATA card.

```
          :
INPUT (c, "DSK")
SELECT INPUT (c)
OPENFILE (c, "MYDAT.DAT")
          :
$DATA MYDAT.DAT
```

This is to ensure that your program finds your data in the disk file under the name that you have assigned to it.

If you let Batch assign a name to your data file, you will not know the name that your data file will have and should therefore assign your data file, without a name, to the card reader. Batch will tell the monitor in this case to look for your data in a disk file when your program wants to read it. The following examples illustrate how to do this.

COBOL Example

```
          .
          .
          .
ENVIRONMENT DIVISION.
INPUT-OUTPUT SECTION.
SELECT SALES, ASSIGN TO CDR.
          .
          .
          .
DATA DIVISION.
FILE SECTION.
FD SALES, LABEL RECORDS ARE OMITTED.
          .
          .
          .
```

FORTRAN Example

To read your data from the card reader, you use the unit number 2 or no unit number, as shown below.

```
        .
        .
        .
READ (2,f), list
        .
        .
        .
$DATA
        .
        .
        .
READ f, list
        .
        .
        .
$DATA
```

ALGOL Example

In an ALGOL program, you would assign the desired channel (signified by c) to the card reader, select the desired channel, but you would not explicitly open the named file on the channel because the file does not have a name that is known to you.

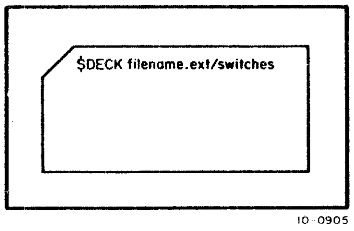
```
        .
        .
        .
INPUT (c, "CDR")
SELECT INPUT (c)
        .
        .
        .
$DATA
```

The $DATA card cannot be used for data for programs written in languages other than ALGOL, COBOL, FORTRAN, and MACRO. It can, however, be used for programs that are in relocatable binary form. Thus, data for BASIC programs cannot be copied by means of the $DATA card; you should instead use the $DECK card, described below.

2.4.4 The $DECK Card

You can put the $DECK card in front of any program, data, or other set of information to make Batch copy the program, data, or information into a disk file. Batch does not insert a command into the control file when it reads the $DECK card. You must include commands in your card deck that Batch will copy into the control file to process the file created by Batch because of the $DECK card.

The form of the $DECK card is:



filename.ext	specifies the optional filename and extension that you can tell Batch to put on the file that it creates for your program or data. If you omit the filename and extension, Batch will create a unique name for your file.
/switches	are switches to Batch to tell it how to read the cards in your deck. The switches are described below.

/WIDTH:n Switch

Normally, Batch reads up to 80 columns on every card in your deck. You can make Batch stop reading at a specific column by means of the /WIDTH switch, in which you indicate the number of a column at which to stop. Thus, if you have information in the last 10 columns of each card in your deck, you can tell Batch to read only up to column 70 by specifying

```
/WIDTH:70
```

on the $DECK card.

/SUPPRESS:OFF Switch

When Batch reads the cards in your deck, it normally does not copy any trailing spaces into the disk file to save space on the disk. If you want Batch to copy everything on the cards up to column 80 or any column that you specify in the /WIDTH switch, you must include the /SUPPRESS:OFF switch on the $DECK card.

Examples

The simplest form of the $DECK card is:

```
$DECK
```

This card causes Batch to copy your deck into a disk file and to assign a unique name to it. All 80 columns of the cards are read and trailing spaces are not copies into the file.

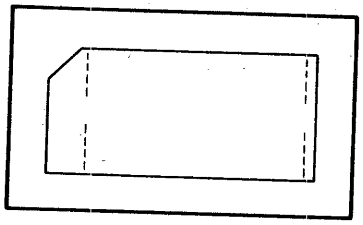
The following shows an example of a $DECK card.

```
$DECK MYDECK.CDS /WIDTH:50 /SUPPRESS:OFF
```

The deck that follows this card is copied into a disk file named MYDECK.CDS. When Batch reads the cards in the deck, it reads up to column 50 and copies trailing spaces into the file.

2.4.5 The End-of-File Card

You must put the end-of-file card at the end of the deck containing your complete job to tell Batch that it has reached the end of your job. Unlike the other Batch control cards, the end-of-file card does not have a dollar sign ($) and a command on it. It contains special punches that are recognized by Batch as the end-of-file. These punches must be in rows 6,7,8, and 9 of column 1. So that the end-of-file card can be recognized in any orientation (e.g., upside down), you should punch rows 12, 11,0,1,6,7,8, and 9 and leave rows 2,3,4, and 5 blank in both columns 1 and 80. If you omit the end-of-file card, an error message will be issued unless the installation makes the operator put the card on any deck that does not have one. However, your job will still be scheduled. The form of the end-of-file card is shown below.

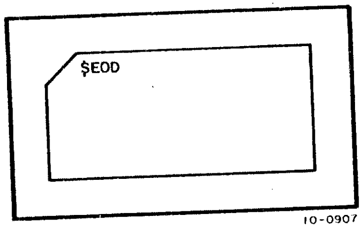


10-0906

2.4.6 The $EOD Card

You put a $EOD card at the end of the cards being copied into a file due to a $DECK, $DATA, or $language card. This card tells Batch to stop copying cards into the file. If another Batch control card follows the cards being copied, you don't need the $EOD card because Batch stops copying cards into a file when it reads a Batch control card. The only time that the $EOD card is necessary is when you wish to follow the cards being copied into a file by a card other than a control card, e.g., a card containing a command. Refer to Section 2.3 for a description of including commands in your deck.

The $EOD card has the form:



An example of using the $EOD card is shown below where the user wishes to load the COBOL debugging program COBDDT with his program.

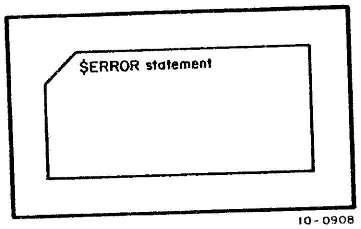
```
$COBOL MYPROG.CBL
       :
$EOD
.LOAD %S MYPROG.CBL, SYS:COBDDT
.START MYPROG
```

If the $EOD card had not been included in the above example, the .LOAD and .START commands would have been copied into the file with the COBOL program, rather than being copied into the control file.

2.4.7 The $ERROR Card

You can use the $ERROR card and the $NOERROR card (described later in this chapter) to specify error recovery in the control file. When Batch reads the $ERROR card, it inserts a special Batch command into the control file, the .IF (ERROR) command. This command will later tell Batch what to do when an error occurs when your job is being processed. How to perform error recovery is described in Section 2.5.

The $ERROR card has the form:



statement	is a command to the monitor, to a system program or a special Batch command such as .GOTO or .BACKTO.

Batch enters an .IF (ERROR) command into the control file when it reads the $ERROR card, and includes the statement from the $ERROR card in the .IF (ERROR) command in the form:

```
.IF (ERROR) statement
```

The Batch commands .GOTO and .BACKTO have the forms:

.GOTO statement label
.BACKTO statement label

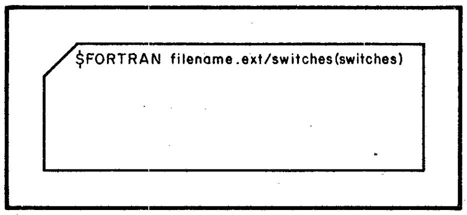
statement label	is the label of a line in the control file. The label can contain from 1 to 6 alphabetic characters and must be followed by a double colon (::) when it is labelling a line.

The .GOTO command tells Batch to search forward in the control file on disk until it finds the line containing the label. The .BACKTO command tells Batch to search back in the control file on disk to find the line containing the label. You must supply the labelled line and any related lines for which Batch will search. Include these lines in your card deck where you want them to be copied into the control file. If Batch cannot find a labelled line that is searching for as a result of a .GOTO or a .BACKTO statement, it terminates your job.

2.4.8 The $FORTRAN Card

You place the $FORTRAN card in front of your FORTRAN program to make Batch copy your program into a disk file and insert a COMPILE command into your control file. Thus, when Batch runs your job, your FORTRAN program will be compiled. You can put some optional information on the $FORTRAN card to tell Batch more about your program or the cards that your program is punched on.

The $FORTRAN card has the form:



10-0909

filename.ext	specifies the optional filename and extension that you can tell Batch to put on the file that it creates for your program. If you omit the filename and extension, Batch will create a unique name for your file and add the extension .F4 to it.
/switches	are switches to Batch to tell it how to read your program. The switches are described below.

(switches)	are switches that Batch passes to the FORTRAN compiler when it puts the COMPILE command in the control file. These switches must be enclosed in parentheses, must not be preceded by slashes, and may or may not be separated by commas. The switches for the FORTRAN compiler are described in Table 11-1 in Chapter 11 of the DECsystem-10 FORTRAN IV Programmer's Reference Manual (DEC-10-AFDO-D).

/WIDTH:n Switch

Normally, Batch reads up to 72 columns on every card of the FORTRAN program. You can make Batch stop reading at a specific column by means of the /WIDTH switch, in which you include the number of the column at which to stop. The FORTRAN compiler only reads FORTRAN statements up to column 72, even if you tell Batch to read up to column 80. But, if you wish to have MPB read only up to column 60, you can specify

```
        /WIDTH:60
```

on the $FORTRAN card.

/SUPPRESS:OFF Switch

When Batch reads the cards of your FORTRAN program, it normally does not copy any trailing spaces into the disk file to save space on the disk. If you want Batch to copy everything on the card up to column 72 or any column that you specify in the /WIDTH switch, you must include the /SUPPRESS:OFF switch on the $FORTRAN card.

/CREF Switch

If you want a cross-reference listing of your FORTRAN program, you can include the /CREF switch on the $FORTRAN card to tell Batch to ask the FORTRAN compiler to produce a cross-reference listing when it compiles your program. This listing is printed as part of your job's output. You do not have to include a command to run the CREF program to get this listing, Batch will do it for you.

/NOLIST Switch

Normally, the $FORTRAN card tells Batch to ask the compiler to generate a compilation listing of your FORTRAN program. The listing is then printed as part of your job's output. If you don't want this listing, you can include the /NOLIST switch on the $FORTRAN card to stop generation of the listing.

Examples

The simplest form of the $FORTRAN card is:

```
        $FORTRAN
```

This card tells Batch to copy your program into a disk file and assign a unique name and the extension .F4. The first 72 columns of the cards are read, trailing spaces are not copied, and a listing file is produced when the job is run. No switches are passed to the compiler. The listing is printed as part of the job's output.

The following is an example of a $FORTRAN card with switches.

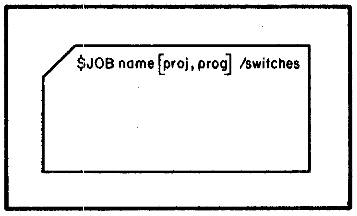
```
$FORTRAN MYPROG.F4 /CREF /NOLIST/SUPPRESS:OFF (I,M)
```

With this card, your FORTRAN program is copied into a disk file named MYPROG.F4 and a COMPILE command is inserted into the control file. The cards are read only up to column 72 and trailing spaces up to column 72 are copied into your file. A cross-reference listing of your program will be generated, but a compilation listing will not. Batch passes the I and M switches to the compiler.

2.4.9 The $JOB Card

You must include the $JOB card as the first card in your deck or as the second card following the $SEQUENCE card, which is described later in this chapter. The $JOB card tells Batch whose job that it is processing and, optionally, the name of the job, and any constraints that you want to place on the job. When Batch reads the $JOB card and the $PASSWORD card, if it is required, it creates the control file and begins the log file for your job. Batch then places commands into the control file that are taken from the cards that follow the $JOB card.

The $JOB card has the form:



10-0910

name	is the optional name that you can give to the job. If you omit the name, Batch will create a unique name for your job. The name of the job is that which Batch gives to your control file and log file. To the job name, Batch adds the extension .CTL for the control file. It adds the extension .LOG to the name for the log file.
[proj,prog]	is your project-programmer number, i.e., the number that you were assigned by the installation to allow you to gain access to the DECsystem-10. Normally, the project-programmer number is two numbers separated by a comma and enclosed in square brackets.

/switches	are switches to Batch to tell it the constraints that you have placed on your job. They are described below.

/AFTER:dd-mmm-yy hhmm Switch

If you don't want Batch to run your job until after a certain time on a certain day, you can include the /AFTER switch on your $JOB card. The date and time are specified in the form dd-mmm-yy hhmm (e.g., 20-MAY-72 0215). If this switch is not included, Batch runs your job at the time that it would normally schedule such a job, based on its size, the amounts of core and time required, and other parameters.

/AFTER:+t Switch

If you don't want Batch to run your job until after a certain number of minutes have elapsed since the job was entered, include this form of the /AFTER switch on the $JOB card. The number of minutes that the job must wait after it has been entered is specified in the form +t (e.g., +15). If this switch is not included, Batch will schedule the job as it normally does.

NOTE

If any of the programs in your job have output to slow-speed devices such as the card punch, the paper-tape punch, the line printer, and the plotter, do not include an ASSIGN command in your job. Batch will take care of this output for you as long as you specify the switches for these devices, which are described below.

/CARDS:nK Switch

If any program in your job has punched card output, you must include the /CARDS switch on the $JOB card to specify the approximate number of cards that your job will punch. Up to a maximum of 10,000 cards can be specified in the form nK or n (e.g., 5K or 5 specifies 5,000 cards). If you do not specify the /CARDS switch, no cards will be punched, even if you want them. If you do not specify enough cards, the remaining output over the number of cards specified will be lost without notification to you.

/CORE:nK Switch

You can specify the amount of core in which the programs in your job will run by means of the /CORE switch. You specify the amount of core in the form n or nK (e.g., 25 or 25K). You should try to estimate as closely as possible the amount of core that your job will need. If you don't specify enough, your job can't run. If you don't specify the amount of core that your job will need, Batch will assume 25K or an amount set by the installation.

/FEET:n Switch

If any program in your job has punched paper-tape output, you must include the /FEET switch on the $JOB card to specify the approximate number of feet of paper tape that your job will punch. You specify the number of feet in the form n (e.g., 50). If you do not specify the /FEET switch, no paper tape will be punched, even if you want it. If you do not specify enough paper tape, the output that remains over the number of feet that you specify will be lost and the message ?OUTPUT FORMS LIMIT EXCEEDED will be punched in block letters in the tape.

/PAGES:n Switch

Normally, Batch allows your job to print up to 100 pages. Included in this number are the log file and any compilation listings that you may request. If you need more than 100 pages for your job, you must include the /PAGES switch on the $JOB card to indicate the approximate number of pages that your job will print. If your output exceeds either the maximum that Batch allows or the number that you specified in the /PAGES switch, the excess output will not be printed and the message ?OUTPUT FORMS LIMIT EXCEEDED will be written in the log file. However, even if you exceed the maximum, the first 10 pages of the log file will be printed.

/TIME:hh:mm:ss Switch

Normally, Batch allows your job to use up to 5 minutes of central processor time. Central processor (CPU) time is the amount of time that your job runs in core, not the amount of time that it takes Batch to process your job. If you need more than 5 minutes of CPU time, you must include the /TIME switch on the $JOB card to indicate the approximate amount of time that you will need. If you don't specify enough time, Batch will terminate your job when the time is up. However, if you specify a large amount of time, Batch may hold your job in the queue until it can schedule a large amount of time for it.

The value in the /TIME switch is given in the form hh:mm:ss (hours:minutes:seconds). However, if you specify only one number, Batch assumes that you mean seconds. Two numbers separated by a colon (:) is assumed to mean minutes and seconds. Only when you specify all three numbers, separated by colons, does Batch assume that you mean hours, minutes, and seconds. For example:

/TIME:25	means 25 seconds
/TIME:1:25	means 1 minute and 25 seconds
/TIME:1:25:00	means 1 hour and 25 minutes

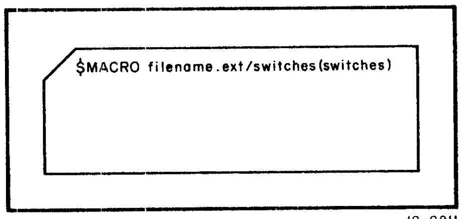
/TPLOT:t Switch

If you have any programs in your job that do output to the plotter, you must include the /TPLOT switch on the $JOB card so that your output will be plotted. If the /TPLOT switch is not included, no output will be plotted. If enough minutes (specified in the form t) are not specified, any plotter output left after the time has expired will be lost without notification to you.

2.4.10 The $MACRO Card

You place a $MACRO card in front of your MACRO program to make Batch copy your program into a disk file and insert a COMPILE command into your control file. Thus, when Batch runs your job, your MACRO program will be assembled. You can put some optional information on the $MACRO card to tell Batch more about your program or the cards that your program is punched on.

The $MACRO card has the form:



```
$MACRO filename.ext/switches(switches)
```

filename.ext	specifies the optional filename and extension that you can tell Batch to put on the file that it creates for your program. If you omit the filename and extension, Batch will create a unique name for your file and add the extension .MAC to it.
/switches	are switches to Batch to tell it how to read your program and the kind of listings that you want. The switches are described below.
(switches)	are switches that Batch passes to the MACRO assembler when it puts the COMPILE command in the control file. The switches must be enclosed in parentheses, must not be preceded by slashes, and may or may not be separated by commas. The switches for the MACRO assembler are described in Table H-1 in Appendix H of the DECsystem-10 MACRO-10 Assembler Programmer's Reference Manual (DEC-10-AMZB-D).

/WIDTH:n Switch

Normally, Batch reads up to 80 columns on every card of your MACRO program. You can make Batch stop reading at a specific column by means of the /WIDTH switch, in which you include the number of the column at which to stop. Thus, if you wish to have Batch read only up to column 70, you can specify

```
/WIDTH:70
```

on the $MACRO card.

/SUPPRESS:OFF Switch

When Batch reads the cards of your MACRO program, it normally does not copy any trailing spaces into the disk file to save space on the disk. If you want Batch to copy everything on the cards up to column 80 or any column that you specify in the /WIDTH switch, you must include the /SUPPRESS:OFF switch on the $MACRO card.

/CREF Switch

If you want a cross reference listing of your MACRO program, you can include the /CREF switch on the $MACRO card to tell Batch to ask the MACRO assembler to produce a cross-reference listing when it assembles your program. This listing is printed as part of your job's output. You do not have to include a command to run the CREF program to get this listing, Batch will do it for you.

/NOLIST Switch

Normally, the $MACRO card tells Batch to ask the assembler to generate an assembly listing of your MACRO program. The listing is then printed as part of your job's output. If you don't want this listing, you can include the /NOLIST switch on the $MACRO card to stop generation of the listing.

Examples

The simplest form of the $MACRO card is:

```
        $MACRO
```

This card tells Batch to copy your program into a disk file and assign a unique name and the extension .MAC to it. All 80 columns of the cards are read, trailing spaces are not copied, and a listing file is produced when the job is run. The listing is printed as part of the job's output. No switches are passed to the assembler.

The following is an example of a $MACRO card with switches.

```
        $MACRO MYPROG.MAC /SUPPRESS:OFF /WIDTH:72 (P,Q,X)
```

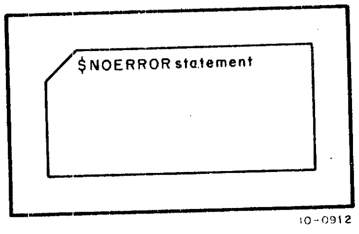
With this card, your MACRO program is copied into a disk file named MYPROG.MAC and a COMPILE command is inserted into the control file. The cards are read only up to column 72 and trailing spaces are copied into your file. An assembly listing is generated, and Batch passes the P,Q, and X switches to the assembler.

2.4.11 The $NOERROR Card

You can use the $NOERROR card and the $ERROR card (described in Section 2.3.7) to specify error recovery in the control file.

When Batch reads the $NOERROR card, it inserts a special Batch command into the control file, the .IF (NOERROR) command. This command tells Batch what to do when an error occurs when your job is being processed. How to perform error recovery is described in Section 2.5.

The $NOERROR card has the form:



statement	is a command to the monitor or a special Batch command such as .GOTO or .BACKTO.

Batch enters an .IF (NOERROR) command into the control file when it reads the $NOERROR card, and includes the statement from the $NOERROR card in the .IF (NOERROR) command in the form:

.IF (NOERROR) statement

The Batch commands .GOTO and .BACKTO have the forms:

.GOTO statement label
.BACKTO statement label

statement label	is the label of a line in the control file. The label can contain from 1 to 6 alphabetic characters and must be followed by a double colon (::) when it is labelling a line.

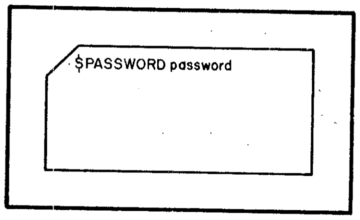
The .GOTO command tells Batch to search forward in the control file until it finds the line containing the label. The .BACKTO command tells Batch to search back in the control file to find the line containing the label. You must supply the labelled line and any related lines for which Batch will search. Include these lines in your card deck where you want them to be copied into the control file. If Batch cannot find a labelled line that is searching for as a result of a .GOTO or a .BACKTO statement, it terminates your job.

2.4.12 The $PASSWORD Card

You put the password that has been assigned to you on the $PASSWORD card to tell Batch that you are an authorized user of the system.

In conjunction with the $JOB card, the $PASSWORD card identifies you to Batch and tells Batch to create the control file and log file for your job. If you put a password on the $PASSWORD card that does not match the password stored in the system for you, Batch will not create any files and will terminate your job. Some installations may not require the $PASSWORD card; if it is required at your installation, you must put it immediately after the $JOB card.

The $PASSWORD card has the form:



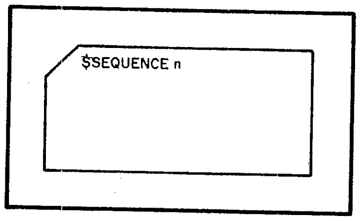
```
$PASSWORD password
```

password	is a 1 to 6 character password that is stored in the system to identify you.

2.4.13 The $SEQUENCE Card

You use the $SEQUENCE card to specify a unique sequence number for your job. This card may or may not be required by the installation or may be supplied by the personnel at the installation. If the card is required, you must include it as the first card in the deck containing your job.

The form of the $SEQUENCE card is:

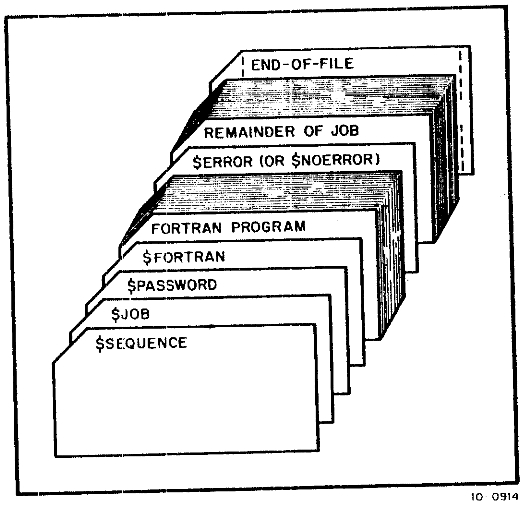


```
$SEQUENCE n
```

n	is the unique sequence number assigned to your job.

2.5 SPECIFYING ERROR RECOVERY IN THE CONTROL FILE

Normally, when an error occurs in your job, Batch terminates the job and, if the error occurred when one of your programs was running, causes a dump of your core area. The dump is printed with your output and log file. You can specify recovery from errors in the control file by means of the $ERROR and $NOERROR cards, described in Sections 2.4.7 and 2.4.11. You must include one of these cards at the point in the control file that an error may occur. When an error occurs, Batch examines the next monitor-level line (i.e., not a line that contains data or a command string to a system program) to find an .IF (ERROR) statement to tell it what to do about the error. If an error does not occur and an .IF (ERROR) statement is present, the .IF (ERROR) statement is not executed. Thus, if you have a program that you are not sure is error-free, you can include a $ERROR or $NOERROR card to tell Batch what to do if an error occurs, as shown in the following example.



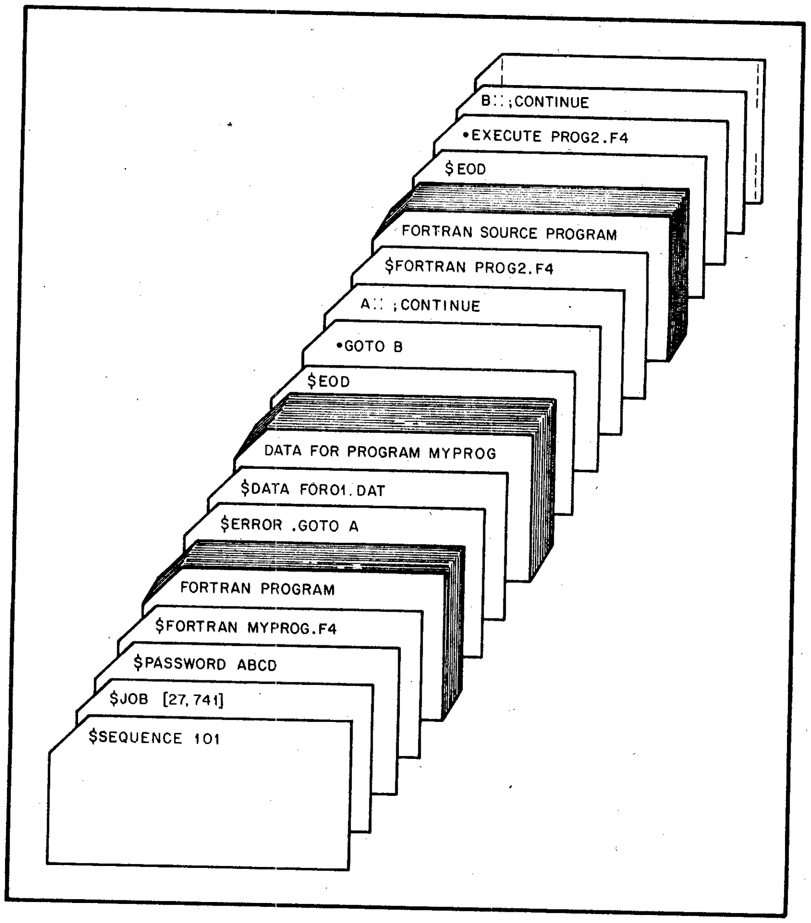
The above cards would cause Batch to make the following entries in the control file.

```
.COMPILE ...
.IF (ERROR) statement
   .
   .
   .
```

On either the $ERROR or $NOERROR card, you must include a statement that tells Batch what to do. You can use any monitor command, a command to a program, or one of the special Batch commands. The .GOTO and .BACKTO commands are two Batch commands for this purpose. Refer to Section 2.4.7 for descriptions of these commands. Be sure, if you use .GOTO or .BACKTO on your $ERROR or $NOERROR card, that you supply a line for the control file that has the label that you specified in the .GOTO or .BACKTO commands.

Two sample jobs are shown below. The first shows using $ERROR and the .GOTO command to specify error recovery. The second example shows the use of the $NOERROR card and the .GOTO command.

If you have a program that you are not sure will compile without errors, you can include another version of the same program in your job (that hopefully will compile) and tell Batch to compile the second program if the first has an error. The cards to enter this job are shown below.



These cards set up the following control file for you.

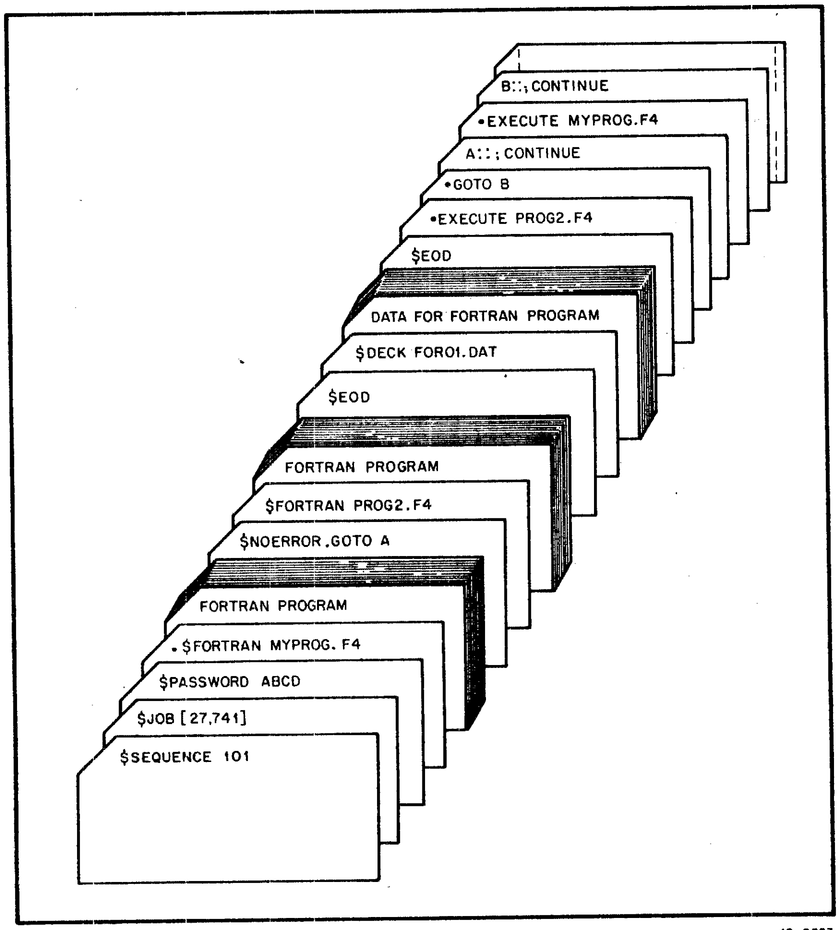
```
.COMPILE /COMPILE MYPROG.F4 /LIST
.IF (ERROR) .GOTO A
.EXECUTE MYPROG.REL /MAP:MAP.LST
.GOTO B
A:: ;CONTINUE
.COMPILE /COMPILE PROG2.F4 /LIST
.EXECUTE PROG2.F4
B:: ;CONTINUE
```

The $FORTRAN card told Batch to copy the program MYPROG.F4 into a disk file and to insert a COMPILE command into the control file. The $ERROR card told Batch to insert .IF (ERROR) .GOTO A into the control file. The data was copied into a disk file and an EXECUTE command was put into the control file because of the $DATA card. The $EOD card told Batch to stop copying cards into the data file, so Batch put the next two lines into the control file. The second $FORTRAN card told Batch to copy the program PROG2.F4 into a disk file and put a COMPILE command into the control file. Another $EOD card told Batch to stop copying into the program file, so Batch put the next two lines into the control file. An EXECUTE command was used instead of a

$DATA card because the data was already in a file on disk, although the $DATA card does not have to have data with it to put an EXECUTE command in the control file.

When the job is started, Batch reads the control file and passes commands to the monitor. If an error occurs in the compilation of the first program, Batch finds the .IF statement and executes the .GOTO command contained in it. The command tells Batch to look for the line labelled A, which contains a comment, so Batch goes on to the next line. The second program is compiled and then executed with the data. The next line is a comment, so Batch continues to the end of the control file. If an error does not occur in the first program, Batch skips the .IF statement, executes the program with the data, skips the unnecessary error procedures, and continues to the end of the control file.

A variation of the above procedure is shown below using the $NOERROR card and the .GOTO command. The difference is that Batch skips the .IF statement if an error occurs, and performs it if an error does not occur.



10-0923

Batch reads the cards and puts the following commands into the control file.

```
.COMPILE /COMPILE MYPROG.F4 /LIST
.IF (NOERROR) .GOTO A
.COMPILE /COMPILE PROG2.F4 /LIST
.EXECUTE PROG2.F4
.GOTO B
A:: ;CONTINUE
.EXECUTE MYPROG.F4
B:: ;CONTINUE
```

The $FORTRAN card tells Batch to copy the FORTRAN program into a file named MYPROG.F4, and to insert a COMPILE command into the control file. The $ERROR card tells Batch to insert an .IF command into the control file. The second $FORTRAN card tells Batch to copy the second program into a disk file named PROG2.F4 and to insert another COMPILE command into the control file. Instead of a $DATA card, a $DECK card is used to tell Batch to copy the data into a disk file named FOR01.DAT. The $DATA card is not used here because it would have the names of both programs in its list for the EXECUTE command generation, which would cause an error when the job is run. To tell Batch to stop copying cards into the data file, the $EOD card comes next. Thus, Batch copies the next five lines into the control file.

When the job is run, Batch passes the COMPILE command to the monitor to compile the first program. If an error does not occur, the .IF command is read and the .GOTO command is executed. Batch skips to the line labelled A, which is a comment, and continues reading the control file. The program MYPROG.F4 is executed with the data and the end of the job is reached. If an error occurs, Batch skips the .IF statement and continues reading the control file. PROG2.F4 is compiled and then executed with the data. Batch is then told to go to the line labelled B, which is a comment line. The end of the job follows.

The examples shown above illustrate only two ways that you can specify error recovery in the control file. You can also use the .BACKTO command or any monitor command that you choose to help you recover from errors in your job.

You do not have to attempt to recover from errors while your job is running. You can correct your errors according to the error messages in the log file when your job is returned to you, and then run your job again. Batch will also print a dump of your core area if an error occurs while your job is running and you have not specified error recovery. If you can read dumps, this can also aid you to correct your errors. The log file and dumps are described in Chapter 4.

CHAPTER 3
ENTERING A JOB TO BATCH FROM A TERMINAL

When you enter a job to Batch from a timesharing terminal, you must create a control file that Batch can use to run your job. The control file contains all the commands that you would use to run your job if you were running under timesharing. For example, if you wanted to compile and execute a program called MYPROG.CBL, the typeout would appear as follows.

```
.COMPILE MYPROG.CBL         (Your request)
COBOL:   MYPROG      }
EXIT                 }        The system's reply
.EXECUTE MYPROG.CBL         (Your request)
LOADING              }
LOADER 1K CORE       }
EXECUTION            }        The system's reply
EXIT                 }
```

The control file to tell Batch to run the same job appears as the following.

```
.COMPILE MYPROG.CBL
.EXECUTE MYPROG.CBL
```

When the job is run, the commands are passed to the monitor to be executed. The commands and their replies from the monitor are written in the log file so that the entire dialog shown above appears in the log file.

To create a control file and submit it to Batch from a terminal, you must perform the following steps.

1. LOGIN to the system as a timesharing user.
2. Write a control file using an editor such as TECO or LINED.
3. When you finish the control file, close and save it on disk.
4. Submit the job to Batch using the monitor command SUBMIT or QUEUE INP:.

You can then wait for your output to be returned at the designated place.

3.1 CREATING THE CONTROL FILE

After you have logged into the system as you normally would to start a timesharing job, you must run an editor so that you can create your control file.

The control file can contain monitor commands, system program commands, data that would normally be entered from a terminal, and special Batch commands. The Batch commands are described in

Section 3.3. What you write in the control file depends on what you wish your job to accomplish. An example of a job that you can enter to Batch from a terminal is as follows.

1. Compile a program that is on disk.
2. Load and execute the program with data from another disk file.
3. Print the output on the line printer.
4. Write the output into a disk file also.
4. Compile a second program.
6. Load and execute the second program with the data output from the first program.
7. Print the output from the second program.

The control file that you would write for the above job is as follows.

```
.COMPILE MYPROG.F4/COMPILE
.EXECUTE MYPROG.F4
.COMPILE PROG2.F4/COMPILE
.EXECUTE PROG2.F4
```

You include statements in your programs to read the data from the disk files and write the output to the printer and the disk. The output to the line printer is written with your log file as part of the total output of your job.

If an error occurs in your job, Batch will not continue, but will terminate the job and, if the error occurs while one of your programs is running, cause a dump to be taken of your core area. The dump is then printed with your job's output. To avoid having your job terminated because an error occurs, you can specify error recovery in the control file using the special Batch commands. Error recovery is described in Section 3.4.

Any monitor command that you can use in a timesharing job can be used in a Batch job with the following exceptions. The ATTACH, DETACH, CCONT, CSTART, and SEND commands have no meaning in a Batch job. If you include one of these commands in your job, Batch will write the command and an error label BAERR into your log file, will not process the command, and will then continue the job from that point. Do not include a LOGIN command in your control file because Batch logs the job for you. If you put in a LOGIN command, your job will be terminated.

3.1.1 Format of Lines in the Control File

Since you can put monitor, system program, and Batch commands, as well as data into the control file, you have to tell Batch what kind of line it is reading. The format of each of these lines is described below. Each line normally begins in column 1, but Batch always starts reading at the first nontab or nonblank character, regardless of the column in which it appears.

To include a monitor or Batch command, you must put a period (.) in the first column and follow it immediately with the command. Any information that follows a monitor command is in the format shown for the command in Chapter 3 of the DECsystem-10 Operating System Commands manual. Any information that follows a Batch command is in the format shown in Section 3.3 in this chapter.

If you include a command string to a system program, you must place an asterisk (*) in column 1 and follow it immediately with the command string. For the format of command strings, refer to the manual for the specific system program that you wish to use.

If you want to include a command to a system program that does not accept carriage return as the end of the line (e.g., TECO and DDT), you must substitute an equal sign (=) for the asterisk so that Batch will suppress the carriage return at the end of the line.

To include data for your program in the control file, write it as you would data that is read from a separate file. The only restriction on data in the control file is that alphabetic data that is preceded by a dollar sign ($) must be preceded by an additional dollar sign so that Batch will not mistake it for a comment.

If you put any special characters other than those described above in the first column of the line, you may get unexpected results because Batch interprets other special characters in special ways. If you want to know about other special characters, refer to Chapter 3 of the DECsystem-10 Operating System Commands manual.

If you have more information than will fit on one line, you can continue on the next line by placing a hyphen (-) as the last nonspace character on the line to be continued and the rest of the information on the next line.

Comments can also be included either as separate lines in the control file or on lines containing other information. To include a comment on a separate line, you must put a semicolon(;) in column 1 and follow it immediately with the comment. To add a comment to a line, you must precede the comment with a semicolon (;) after all the information that you need has been put on the line.

3.2 SUBMITTING THE JOB TO BATCH

After you have created the control file and saved it on disk, you must enter it into the Batch queue so that it can be run. All programs and data that are to be processed when the job is run must be made up in advance or be generated during the running of the job. You can have them on any medium but, if they are on devices other than disk, you must include commands in your control file to have the operator mount the devices on which your programs and data reside. It is recommended that your programs and as much of your data as is possible reside on disk. An example of including MOUNT commands in the control file to mount tapes is shown in Chapter 5.

You enter your job into Batch's queue by means of the SUBMIT or QUEUE INP: monitor command. These commands have the forms:

```
SUBMIT jobname = control filename.ext, log filename.ext /switches
QUEUE INP:jobname = control filename.ext, log filename.ext / switches
```

jobname	is the name that you give to your job. If this name is omitted, Batch uses the name of the control file.
control filename.ext	is the name that you have given to the control file that you created. You can add an extension, but if you don't, Batch will assume an extension of .CTL.
log filename.ext	is the name that Batch will give the log file when it is created. You can add an extension, but if you don't, Batch will assume an extension of .LOG.

You must specify the name of the control file. If the name of the log file is omitted, its name will be taken from the name of the control file.

/switches	are switches to Batch to tell it how to process your job and what your output will look like. Most switches can appear anywhere in the command string; however, a few must be placed after the files to which they pertain. The various kinds of switches are described below.

Three kinds of switches are available to you to use in the SUBMIT and QUEUE INP: commands. The switches are: queue operation, general, and file control. Each category of switch and the switches in each category are described in the following sections.

3.2.1 Queue Operation Switches

Queue operation switches describe the actions that you want Batch to perform with your job. Only one of this type of switch can be placed in the command string, and it can appear anywhere in the command string.

/CREATE Switch

With the /CREATE switch, you tell Batch that you are entering a job into its queue. The job will then wait in the queue until Batch is ready to process it. If you omit a queue operation switch from the SUBMIT command string, Batch will assume the /CREATE switch, i.e., it will assume that you are entering a job. An example of this switch follows.

```
SUBMIT MYJOB = MYFILE,CTL, MYLOG.LOG /CREATE
```

/KILL Switch

You put the /KILL switch in a SUBMIT command to tell Batch that you want to delete a job that you previously entered into its queue. For example, if you submit a job and discover that you left a command out of the control file, you could delete the queue entry by issuing another SUBMIT command for that job with a /KILL switch in it. After you have corrected the control file, you could resubmit the job to Batch. However, if Batch has already started to run your job, it will ignore your request to delete the job and issue the message %QUEUE REQUEST INP:jobname [proj,prog] INTERLOCKED IN QUEUE MANAGER. When you use the /KILL switch, you must specify the job name in the SUBMIT command or you will kill all the jobs that you may have in the Batch input queue.

/MODIFY Switch

If you want to change any switch value that you have previously entered in a SUBMIT command, you can include the /MODIFY in a new SUBMIT command to tell Batch which switch value that you want to change. You can change any switch value that can be entered in a SUBMIT command. The switch value that you want changed is written immediately after the /MODIFY switch. For example, to change the number of pages in a /PAGE switch (described below), you could issue the following command:

```
        SUBMIT MYJOB = /MODIFY/PAGE:500
```

The value specified in the /PAGE switch that follows the /MODIFY switch replaces the previous value. If Batch has already started the job in which you wish to change a switch, the /MODIFY switch will be ignored, and Batch will issue the message %QUEUE REQUEST INP:jobname [proj,prog] INTERLOCKED IN QUEUE MANAGER.

3.2.2 General Switches

You use the general switches to define limits for your job. Such limits as core, pages of output, and the time that your job will run can be specified as general switches. Each general switch can be specified only once in a SUBMIT command, although each can be modified in subsequent SUBMIT commands by means of the /MODIFY switch. You can put a general switch anywhere in the command string because it affects the entire job, not just one file in the job.

/AFTER:t Switch

If you don't want Batch to run your job until after a certain time or until after a certain number of minutes have elapsed since the job was entered, you can include the /AFTER switch in the SUBMIT command string. The time is specified in the form hhmm (e.g., 1215) and the number of minutes that the job must wait is specified in the form +t (e.g., +15). If you omit the switch, or the colon and the value in the switch, Batch will schedule your job as it normally would.

NOTE

If any of the programs in your job have output to slow-speed devices such as the card punch, the paper-tape punch, the line printer, and the plotter, do not include an ASSIGN command in your job. Batch will take care of this output for you as long as you specify the switches for these devices, which are described below.

/CARDS:n Switch

If any program in your job has punched card output, you must include the /CARDS switch in the SUBMIT command to specify the approximate number of cards that your job will punch. The number of cards is specified in the form n (e.g.; 1000). If you do not specify the /CARDS switch, no cards will be punched, even if you want them. If you specify the switch without the colon and a value, up to 2000 cards can be punched by your job. If you do not specify enough cards, the output that remains after the limit is reached will be lost without notification to you.

/CORE:n Switch

You can specify the maximum amount of core in which the programs in your job will run by means of the /CORE switch. You specify the amount of core in the form n (e.g., 25) which indicates decimal thousands. You should try to estimate as closely as possible the amount of core that your job will need. If you don't specify enough, you job can't run to completion. If you omit the switch, Batch will assume 25K of core or an amount set by the installation. If you specify the switch without the colon and a value, Batch will assume 40K of core or an amount set by the installation.

/FEET:n Switch

If any program in your job has punched-paper-tape output, you must include the /FEET switch in the SUBMIT command to specify the approximate number of feet of paper tape that your job will punch. You specify the number of feet in the form n (e.g., 50). If you do not specify the /FEET switch, no paper tape will be punched, even if you want it. If you specify the /FEET switch without the colon and a value, Batch will assume the number of feet equal to 10 times the number of disk blocks that your paper tape output would occupy plus 20. If you do not specify enough paper tape, the output that remains after the limit is exceeded will be lost and the message ?OUTPUT FORMS LIMIT EXCEEDED will be punched into the tape in block letters.

/PAGE:n Switch

Normally, Batch allows your job to print up to 200 pages. Included in this number are the log file and any listings that you may request. If you need more than 200 pages for your job, you must include the /PAGES switch in the SUBMIT command to indicate the approximate number of pages that your job will print. If you include the switch without the colon and a value, Batch will assume that you will print up to 2000 pages. If your output exceeds either the maximum that Batch allows or the number that you specified in the /PAGE switch, the excess output will be lost and the message ?OUTPUT FORMS LIMIT EXCEEDED will be printed. However, even if you exceed the maximum, the first 10 pages of the log file will be printed.

/TIME: hh:mm:ss Switch

Normally, Batch allows your job to use up to 5 minutes of central processor time. Central processor (CPU) time is the amount of time that your job runs in core, not the amount of time that it takes

Batch to process your job. If you need more than 5 minutes of CPU time, you must include the /TIME switch in the SUBMIT command to indicate the approximate amount of time that you will need. If you specify the switch without the colon and a value, Batch will assume that you need 1 hour of CPU time. If you don't specify enough time, Batch will terminate your job when the time is up.

The value in the /TIME switch is given in the form hh:mm:ss (hours:minutes:seconds). However, if you specify only one number, Batch assumes that you mean seconds. Two numbers separated by a colon is assumed to mean minutes and seconds. Only when you specify all three numbers, separated by colons, does Batch assume that you mean hours, minutes, and seconds. For example:

/TIME:25	means 25 seconds
/TIME:1:25	means 1 minute and 25 seconds
/TIME:1:25:00	means 1 hour and 25 minutes.

/TPLOT:t Switch

If you have any programs in your job that do output to the plotter, you must include the /TPLOT switch in the SUBMIT command so that your output will be plotted. If the /TPLOT switch is not included, no output will be plotted. If you specify the switch without the number of minutes (specified in the form t), Batch will allow output equal to 10 minutes of plotter time. If enough minutes are not specified, any plotter output left after the time has expired will be lost without notification to you.

3.2.3 File-Control Switches

File-control switches allow you to specify parameters for individual files in the SUBMIT command. The control file can receive a special parameter, while the log file does not, and vice versa. If you place a file-control switch before the two filenames in the SUBMIT command, the switch applies to both files in the request. If you place the switch after one of the files in the command, it refers to only that file.

/DISPOSE Switch

The /DISPOSE switch can have one of three values:

```
/DISPOSE:DELETE
/DISPOSE:PRESERVE
/DISPOSE:RENAME
```

/DISPOSE:DELETE allows you to specify that either the control file or the log file (or both) should be deleted after the job is run. The log file is deleted from your disk area only after it has been printed.

/DISPOSE:PRESERVE allows you to specify that one or both of your files should be left in your disk area after the job is finished and all output printed.

/DISPOSE:RENAME tells Batch that you want the specified file to be taken from your disk area immediately and put in Batch's disk area. In the case of the log file, /DISPOSE:RENAME only works for a log file that already exists on your disk area. Do not use /DISPOSE:RENAME for a log file that does not yet exist. After the job has been run and the output has been printed, the file that was renamed is deleted from Batch's disk area.

If you omit the /DISPOSE switch, Batch assumes /DISPOSE:PRESERVE. That is, both the control file and the log file are saved in your disk area. If you plan to use the control file again, then it is best to omit the /DISPOSE switch for the control file. If you don't want to keep the control file because you don't have enough room in your disk area, specify either /DISPOSE:DELETE or /DISPOSE:RENAME. /DISPOSE:DELETE will cause the control file to stay in your disk area until after the job is finished and then be deleted. /DISPOSE:RENAME will cause Batch to immediately move your control file to its own disk area where it will stay until the job is finished, at which time it will be deleted. You should use /DISPOSE:RENAME when you will be over your logged-out quota if the control file remains in your disk area when you log off the system.

Unless you have some use for the copy of the log file that will remain in your disk area even after it has been printed, use the /DISPOSE:DELETE switch to have the log file deleted after it is printed. If you do not delete the log file and you run the job again using the same log filename, your new log file will be appended to the old log file and they will both be printed as part of the new job.

The switches, and the assumptions made if they or their values are omitted, are all subject to change by each installation. Check with the installation where you run your jobs to find out what differences exist between the values described here and those at the installation. Additional switches are available for use with the SUBMIT command. For information about these switches, refer to the SUBMIT command in Chapter 2 of the DECsystem-10 Operating System Commands manual (DEC-10-MRDC-D). You can obtain further information about Batch in Chapter 3 of the aforementioned manual.

3.2.4 Examples of Submitting Jobs

The following are sample jobs that are entered to Batch by means of the SUBMIT command. The jobs are shown in the following order.

1. Creating the control file.
2. Submitting the job to Batch using the SUBMIT command.

```
.COMPILE MYPROG.F4 /LIST/COMPILE
.EXECUTE MYPROG.F4
```

After the control file to compile and execute the FORTRAN program has been written and saved, you must submit the job to Batch.

```
SUBMIT MYFILE
```

When Batch reads this SUBMIT command, it assumes the following:

1. The control filename and extension are MYFILE.CTL.
2. The name of the job is MYFILE.
3. The log file will be named MYFILE.LOG.
4. Both the control file and the log file will be saved in your disk area (/DISPOSE:PRESERVE).
5. An entry is being created in Batch's queue (/CREATE).
6. No cards will be punched by the job (/CARDS:0).
7. The maximum amount of core to be used to run the job is 25K (/CORE:25).
8. No paper tape will be punched (/FEET:0).
9. 200 is the maximum number of pages to be printed (/PAGE:200).
10. The maximum amount of CPU time is 5 minutes (/TIME:5:00).
11. No plotter time will be used (/TPLOT:0).

The next example shows the control file that was created at the beginning of this chapter being submitted to Batch.

```
.COMPILE MYPROG.F4/COMPILE
.EXECUTE MYFILE.F4
.COMPILE PROG2.F4/COMPILE
.EXECUTE PROG2.F4
```

After you have saved the control file, you must submit the job to Batch.

```
SUBMIT = MYFILE.MYFILE.LOG/DISPOSE:DELETE/TIME:20:00/CARDS:500
```

When Batch reads this request, it assumes the following:

1. The name of the job is MYFILE.
2. The name of the control file is MYFILE.CTL.
3. The log file will be named MYFILE.LOG.
4. An entry is being created in Batch's queue (/CREATE).
5. The log file will be deleted after it is printed (/DISPOSE:DELETE).
6. The control file will be saved in your disk area (/DISPOSE:PRESERVE).
7. A maximum of 500 cards can be punched by the job (/CARDS:500).
8. The maximum amount of core that can be used is 25K (CORE:25).
9. No paper tape will be punched by the job (/FEET:0).
10. 20 is the maximum number of pages that can be printed (/PAGE:20).
11. The maximum amount of CPU time that the job can use is 20 minutes (/TIME:20:00).
12. No plotter time will be used (/TPLOT:0).

If you made an error in the SUBMIT command when you submitted either of these jobs, Batch will type an error message on your terminal to explain your error so that you can correct it.

3.3 BATCH COMMANDS (IN ALPHABETICAL ORDER)

You can write certain special Batch commands in the control file to tell Batch how to process your control file. Each of these commands must be preceded by a period so that Batch will recognize it. The commands are described in detail in the following sections.

3.3.1 The .BACKTO Command

You can use the .BACKTO command to direct Batch to search back in the control file for a line with a specified label. The .BACKTO command has the form:

.BACKTO label

label	is a 1- to 6-character alphanumeric label for a statement. It must be followed by a double colon (::) when it labels a statement to show that it is label.

Normally, Batch reads the control file line-by-line and passes the commands and data to the monitor and your program. When you put a .BACKTO command into the control file, you tell Batch to interrupt the normal reading sequence and to search back in the control file to find a line containing the label specified in the .BACKTO command. When it reaches the labelled line, Batch executes the line and continues from that point (unless the line contains another .BACKTO command or a .GOTO command, described below).

If Batch cannot find the labelled line, it terminates your job. An example of the .BACKTO command is as follows.

```
                 .
                 .
                 .
ABC:: .DIRECT
                 .
                 .
                 .
.BACKTO ABC
```

3.3.2 The .ERROR Command

With the .ERROR command, you can specify to Batch the character that you wish to be recognized as the beginning of an error message. Normally, when Batch reads a message that begins with a question mark (?), it assumes a fatal error has occurred and terminates the job, unless you have specified error recovery (refer.to Section 3.4). If you wish Batch to recognize another character as the beginning of a fatal error message, you must specify the character in the .ERROR command. This command has the form:

.ERROR character

character	is a single ASCII character that is recognized in the DECsystem-10.

If you do not specify a character in the .ERROR command, Batch uses the standard error character, the question mark. When a line that is preceded by the character that you specify in the .ERROR command is passed to Batch from the monitor, a system program or is issued by Batch itself, Batch treats the line as a fatal error and terminates the job, exactly as it would if the line were preceded by a question mark. Any messages preceded by other characters will not be recognized by Batch as errors. The only exception is the ?TIME LIMIT EXCEEDED message. No matter what character you specify as the beginning of an error, Batch will recognize this message and terminate your job.

If you do not include the .ERROR command in your control file, Batch will recognize the question mark as the beginning character of a fatal error message, unless you include the .NOERROR command in your control file to cause Batch to ignore fatal errors (refer to Section 3.3.5).

An example of the .ERROR command follows.

```
                 .
                 .
                 .
        .ERROR   %
                 .
                 .
                 .
        .ERROR
```

In this example, you specify in the middle of the control file that you want Batch to recognize the percent sign (%) as the beginning character of a fatal error from that point in the control file. Further on in the control file, you tell Batch to go back to recognizing the question mark as the beginning of a fatal error message.

3.3.3 The .GOTO Command

You can include the .GOTO command in your control file to direct Batch to skip over lines in the control file to find a specific line. The .GOTO command has the form:

.GOTO label

label	is a 1- to 6-character alphanumeric label for a statement. It must be followed by a double colon (::) when it labels a statement to show that it is a label.

When Batch encounters a . GOTO command in the control file, it searches forward in the control file to find the label specified in the .GOTO command. Batch then resumes processing of the control file at the line with the specified label. If Batch cannot find the labelled line, it terminates your job.

If you do not include a .GOTO command in the control file, Batch reads the control file sequentially from the first statement to the last, unless you include a .BACKTO statement (refer to Section 3.3.1).

An example of the .GOTO command follows.

```
              .
              .
              .
        .GOTO ABC
              .
              .
              .
        ABC:: .DIRECT
```

You can use the .GOTO command as the statement in an .IF command (refer to Section 3.3.4) to aid you in error processing. For example:

```
              .
              .
              .
        .IF (ERROR) .GOTO ABC
              .
              .
              .
        ABC:: .TYPE MYPROG
```

3.3.4 The .IF Command

You can include the .IF command in your control file to specify an error recovery procedure to Batch or to specify normal processing if an error does not occur. The .IF statement has the forms:

.IF (ERROR) statement
.IF (NOERROR) statement

statement	is a command to the monitor, to a program, or to Batch.

The .IF command can be used in two ways as shown in its two forms. You can include the .IF (ERROR) command in your control file at the place where you may have an error. The .IF (ERROR) command must be the next monitor-level line (as opposed to a line in your program or a line of data) in your control file after an error occurs so that Batch will not terminate your job. In the .IF (ERROR) command, you direct Batch to either go back or forward in your control file to find a line that will perform some task for you, or direct Batch to perform a task for you at that point in your control file, or to direct the monitor or any other program to perform some task for you.

You can use the .IF (NOERROR) command also to direct Batch or the monitor to perform tasks for you when an error does not occur at the point in your control file where you place the .IF (NOERROR) command. Thus, if you expect that an error will occur in your program, you can include an .IF (NOERROR) command to direct Batch in case the error does not occur, and then put the error processing lines immediately following the command. Refer to Section 3.4 for an example of using .IF (NOERROR) and .IF (ERROR).

If an error occurs and Batch does not find an .IF command as the next monitor-level line in the control file, Batch writes an error message in the log file and terminates the job. If one of your programs is running when an error occurs and there is no .IF command, Batch causes dump to be taken and terminates your job.

3.3.5 The .NOERROR Command

You can use the .NOERROR command to tell Batch to ignore all error messages issued by the monitor, system programs, and Batch itself. The only exception is the message ?TIME LIMIT EXCEEDED. Batch will always recognize this as an error message and terminate your job. The .NOERROR command has the form:

```
.NOERROR
```

When Batch reads the .NOERROR command, it ignores any error messages that would normally cause it to terminate your job. However, Batch still writes the error message in the log file so that you can examine your errors when your output is returned.

You can use .NOERROR commands in conjunction with .ERROR commands in the control file to control error reporting. For example, if you wish to ignore errors at the beginning and end but not in the middle of the control file, place .ERROR and .NOERROR commands at the appropriate places in the control file. In addition, you can also specify which messages must be treated as fatal errors.

```
       .
       .
       .
.NOERROR
       .
       .
       .
.ERROR %
       .
       .
       .
.ERROR
       .
       .
       .
.NOERROR
```

The first command tells Batch to ignore all errors in your job. The second command tells Batch to recognize as errors any message that starts with a percent sign (%). You change the error reporting with the next command to tell Batch to go back to recognizing messages that begin with a question mark as fatal. The second .NOERROR command tells Batch to ignore all error messages again. If the ?TIME LIMIT EXCEEDED message is issued at any time, Batch will print the message and terminate the job.

3.4 SPECIFYING ERROR RECOVERY IN THE CONTROL FILE

If you don't specify error recovery when an error occurs in your job, Batch terminates the job and, if the error occurs when one of your programs is running, causes a dump of your core area. You can specify error recovery in the control file by means of the Batch commands, especially the .IF command. You must include the .IF command at the point between programs in the control file that an error may occur. When an error occurs, Batch examines the next monitor-level line (i.e., not a line that contains data or a command string to a system program) to find an .IF command to tell it what to do with the error. If an error does not occur and an .IF (ERROR) command is present, the .IF (ERROR) command is not executed. Similarly, if an error does not occur and you have included an .IF (NOERROR) command, the .IF command is processed. Thus, if you have a program that you are not sure is error-free, you can include an .IF command to tell Batch what to do if an error occurs, as shown in the following example.

```
.COMPILE MYPROG.F4
.IF (ERROR) STATEMENT
          .
          .
          .
```

In either the .IF (ERROR) or the .IF (NOERROR) command, you must include a statement that tells Batch what to do. You can use any monitor command or one of the Batch commands. The .GOTO and .BACKTO commands are commonly used for this purpose. Refer to Sections 3.3.1 and 3.3.3 for descriptions of these commands. Be sure, if you use .GOTO or .BACKTO in the .IF command, that you supply a line in the control file that has the label that you specified in the .GOTO or .BACKTO command.

Two sample jobs are shown below. The first shows the .IF (ERROR) command and the .GOTO command to specify error recovery. The second example shows the use of the .IF (NOERROR) and .GOTO commands.

If you have a program that you are not sure will compile without errors, you can include another version of the same program in your job (that hopefully will compile) and tell Batch to compile the second program if the first has an error. You write the control file as follows.

```
.COMPILE /COMPILE MYPROG.F4 /LIST
.IF (ERROR) .GOTO A
.EXECUTE MYPROG.F4
.GOTO B
A:: ;CONTINUE
.COMPILE /COMPILE PROG2.F4 /LIST
.EXECUTE PROG2.F4
B:: ;CONTINUE
```

When the job is run, Batch reads the control file and passes commands to the monitor. If an error occurs in the compilation of the first program, Batch finds the .IF (ERROR) command and executes the .GOTO command contained in it. The command tells Batch to look for the line labelled A, which contains a comment, so Batch continues to the end of the control file. If an error does not occur in the first program, Batch skips the .IF (ERROR) command, executes the program with its data, skips the unnecessary error procedures, and continues to the end of the control file.

A variation of the above procedure is shown below using the .IF (NOERROR) command and the .GOTO command. The difference is that Batch skips the .IF (NOERROR) command if an error occurs, and performs it if an error does not occur. The following is the control file that you would create.

```
.COMPILE /COMPILE MYPROG.F4 /LIST
.IF (NOERROR) .GOTO A
.COMPILE /COMPILE PROG2.F4 /LIST
.EXECUTE PROG2.F4
.GOTO B
A:: !CONTINUE
.EXECUTE MYPROG.F4
B:: !CONTINUE
```

When the job is run, Batch passes the COMPILE command to the monitor to compile the first program. If an error does not occur, the .IF (NOERROR) command and the .GOTO command are executed. Batch skips to the line labelled A, which is a comment, and continues reading the control file. The program MYPROG.F4 is executed with its data and the end of the job is reached. If an error occurs, Batch skips the .IF (NOERROR) command and continues reading the control file. PROG2.F4 is compiled and then executed with the same data that the first program would have used. Batch is then told to go to the line labelled B, which is a comment line. The end of the job follows.

The examples shown above illustrate only two ways that you can specify error recovery in the control file. You can also use the other Batch commands, or any monitor command that you choose to help you recover from errors in your job.

You do not have to attempt to recover from errors while your job is running. You can correct your errors according to the error messages in the log file when your job is returned to you, and then run your job again. Batch will also print a dump of your core area if an error occurs while your job is running and you have not specified error recovery. If you can read dumps, this can also aid you to correct your errors. The log file and dumps are described in Chapter 4.

CHAPTER 4
INTERPRETING YOUR PRINTED OUTPUT

You can receive three kinds of printed output from your Batch job:

1. Output that you request, i.e., the results of your job.
2. Output from Batch, i.e., the log file.
3. Output that is the result of actions by your job or by Batch, the monitor, or system programs. Examples of this output are compilation listings, cross-reference listings, error messages, and core dumps requested by Batch.

4.1 OUTPUT FROM YOUR JOB

Although this chapter deals mainly with printed output, you can have output to any device that the installation supports, as long as the installation allows you to use these devices. If your output is directed to the line printer, it will be printed separate from the log file. The printed output from each program will be preceded by two pages containing your name and project-programmer number and other pertinent information. Following these pages are two header pages containing the name of your output file in block letters. The output follows these header pages. A trailer page follows your output. This page contains the same information that is on the first two pages. The header and trailer pages also include three rows of numbers (read vertically from 001 to 132).

If your output is that which would normally be sent to the terminal, it will be printed in the log file. In the sample output shown in Section 4.4, the output from the program is included in the log file because it is directed to the terminal rather than the line printer.

4.2 BATCH OUTPUT

The output from Batch consists of a log file that contains all the statements in the control file, commands sent to the monitor from Batch for you, and the replies to the commands from the monitor and system programs like the compilers. Any error message sent from the monitor or a system program, or from Batch itself, is also written in the log file. Refer to the <u>DECsystem-10 Operating System Commands</u> manual (DEC-10-MRDC-D) for a list of the error messages from the monitor. The messages from each system program are listed in the applicable manuals.

You can ignore most of the information in the log file because it is system information and need not concern you. If you wish, you can keep it for reference by system programmers if unexpected results occur in your job.

4.3 OTHER PRINTED OUTPUT

Other output that you can get as a result of your job includes compiler and cross-reference listings, loader maps for programs that were successfully loaded, and dumps that you can request or that Batch gives to you when an error occurs in your program.

The compiler and cross-reference listings are those listings generated by the compiler if you request them. When you enter your job from cards, Batch requests compilation listings for you unless you specify otherwise. Cross-reference listings are generated for you only if you specifically ask Batch for them. When you enter your job from a terminal, you must request the listings in the COMPILE command. Also, if you request a cross-reference listing, you must run the CREF program (by means of the CREF command) to get your listing printed.

If you enter your job from cards and include a $DATA card to request execution of a program, Batch requests a loader map for you. This map shows the locations in memory into which your program was placed. If you enter your job from a terminal, you must request a loader map in the EXECUTE command if you wish to have one. If you wish to know the locations into which your program was loaded, the loader map can be of use to you. Otherwise, you can ignore it. A loader map is shown in the sample output in Section 4.4, however, it is not interpreted in this manual.

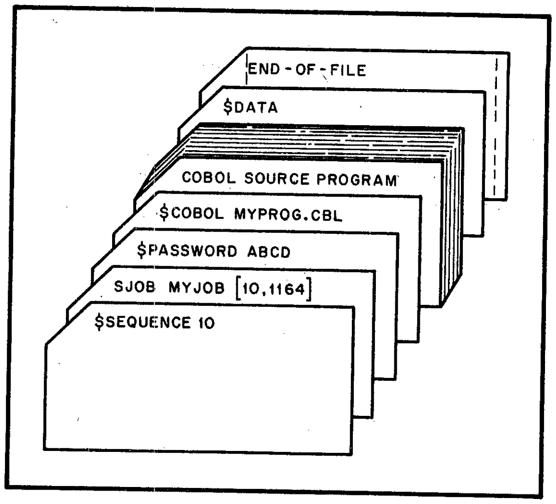
If a fatal error occurs in a program in your job and you have not included an error recovery command to Batch, Batch will not try to recover from the error for you. Instead, it will write the error message in the control file, request dump of your memory area, and terminate your job. The dump is then printed with your output. If you can read dumps, the dump that Batch requests for you may be helpful in finding your errors. Otherwise, you can ignore the dump and use the error messages to locate the errors in your program. A sample dump is shown in Section 4.4, but it is not interpreted. It is shown so that you can recognize it if you ever receive one.

4.4 SAMPLE BATCH OUTPUT

Two sample jobs and their output are shown in the following sections. The first shows a job entered from cards, the second shows a job entered from a terminal. The log file is somewhat different for the two types of jobs. Following the sample jobs is a sample dump.

4.4.1 Sample Output from a Job on Cards

This example shows a job in which a small COBOL program is compiled and executed. The card deck is as follows.



The COBOL program is as follows.

```
IDENTIFICATION DIVISION.
PROGRAM-ID. MYPROG.
ENVIRONMENT DIVISION.
DATA DIVISION.
PROCEDURE DIVISION.
START.
DISPLAY "THIS IS TO SHOW SAMPLE OUTPUT FROM MPB.".
DISPLAY "THESE TWO LINES ARE OUTPUT FROM THE PROGRAM.".
STOP RUN.
```

When the job is run, the program is compiled and a compilation listing is produced. The listing is shown below. Note that the compiler put sequence numbers on the program even though they were not in the original program.

```
P R O G R A M   M Y P R O G,              COBOL 3(43)     21-MAR-72  10:42

0001    IDENTIFICATION DIVISION.
0002    PROGRAM-ID. MYPROG.
0003    ENVIRONMENT DIVISION.
0004    DATA DIVISION.
0005    PROCEDURE DIVISION.
0006    START.
0007    DISPLAY "THIS IS TO SHOW SAMPLE OUTPUT FROM MPB.".
0008    DISPLAY "THESE TWO LINES ARE OUTPUT FROM THE PROGRAM.".
0009    STOP RUN.

NO ERRORS DETECTED
```

After the program is compiled, it is loaded and executed. Since Batch requests a loader map when it puts the EXECUTE command in the control file, the loader map is the next thing printed from your job. It is shown below. Note that each of these print-outs are preceded by headers, which are not shown in these examples.

```
ØØ1246 IS THE LOW  SEGMENT BREAK

MAP        STORAGE MAP     1Ø:42 21-MAR-72

STARTING ADDRESS   ØØ12ØØ   PROG COBOL    FILE MYPROG

.COMM.     ØØØ14Ø    ØØ1Ø4Ø


MYPROG.    ØØ12ØØ    ØØ11Ø3
           START.    ØØ12ØØ       FILES.  ØØØ14Ø       USES.        ØØØ141      SEGWD.     ØØØ142
           ALTER.    ØØØ143       OVRFN.  ØØØ144       POINT.       ØØØ145      COMMA.     ØØØ146
           MONEY.    ØØØ147       MEMRY.  ØØØ15Ø       TRAC1.       ØØØ152      TRAC2.     ØØØ153
           TRAC3.    ØØØ154       %NM.    ØØØ155       %DT.         ØØØ156      %PR.       ØØØ157

TRACED     ØØ1243    ØØØØØ3
           BTRAC.    ØØ1244       PTFLG.  ØØ1245       TRACE.       ØØ1243      TRPD.      ØØ1244
           TRPOP.    ØØ1244

COBOL 1K CORE, 345 WORDS FREE
LOADER USED 2+4K CORE
```

Following loading, the program is executed. The program in this example does not have output to the line printer, instead its output is written to a terminal. Because this is a Batch job, the terminal output is written in the log file. The log file is printed next because the end of the job is reached. The log file contains all the dialog between your job and the monitor and system programs, and some commands that Batch sent to the monitor for you. An annotated log file is shown on the following pages. Note that each line in the log file is preceded by the time of day when the line was written. Following the time is a word that describes what kind of information is on each line. You do not need to know what each of these words means because much of the information is system information.

```
10:41:43 DATE    21-MAR-72     554A1E DUAL CPU   CDRSTK VER 12(17) DSK
10:41:43 CARD    $JOB MYJOB [10,1164]
10:41:44 STSUM   END OF FILE AFTER 15 CARDS, 04 FILES, 03 BLKS

10:41:50 BVERS   BATCON 7(35) INP: SUBJOB 01 OF 06
10:41:50 BDATE   21-MAR-72
10:41:50 BASUM   MYJOB[10,1164] FOR **[10,1164] LOG FILE IN [10,1164]
                 REQUEST CREATED AT  10:41:00 21-MAR-72
                 UNIQUE: 2 RESTART: 1
```

This is system information that Batch enters. It need not concern you.

```
10:41:50 MONTR
10:41:5  MONTR   .LOGIN 10/1164
10:42:01 USER    JOB 24     554A1E DUAL CPU TTY102
10:42:01 USER    OTHER JOBS SAME PPN
10:42:01 USER    1041    21-MAR-72         TUE
10:42:01 MONTR
```

Batch logs your job into the system. The information that follows it is the system response.

```
10:42:01 MONTR   .SET TIME 300.
10:42:08 MONTR
10:42:08 MONTR   .SET SPOOL ALL
10:42:08 MONTR
10:42:08 MONTR
```

These are commands that Batch entered for you.

```
10:42:08 MONTR   .
                 $SEQUENCE 10
                 $JOB MYJOB [10,1164]
                 $COBOL MYPROG.CBL
```

These are the cards that you entered.

```
10:42:08 MONTR   .COMP /COMPILE MYPROG.CBL/LIST   ;CREATED BY CDRSTK
```

This is the command entered by Batch for you.

```
10:42:10 USER    COBOL: MYPROG
10:42:32 MONTR   EXIT
10:42:32 MONTR
10:42:32 MONTR
```

The answer to the COMPILE command from the monitor.

```
                   $DATA                                                    }  Your $DATA card.
1Ø:42:32 MONTR     .SET CDR QAA.CDR                   ;CREATED BY CDRSTK
1Ø:42:32 MONTR                                                                 
1Ø:42:32 MONTR     ..EXEC /MAP:MAP.LPT /REL MYPROG.REL     ;CREATED BY CDRSTK }  Commands entered by Batch for you.
1Ø:42:59 USER      LOADING
1Ø:43:ØØ USER      ØØ1246 IS THE LOW  SEGMENT BREAK
1Ø:43:ØØ USER                                                                    Monitor response to the EXECUTE
1Ø:43:ØØ USER      COBOL 1K CORE                                                 command.
1Ø:43:ØØ USER      EXECUTION
1Ø:43:ØØ USER      THIS IS TO SHOW SAMPLE OUTPUT FROM MPB.                       This is the output from your program.
1Ø:43:ØØ USER      THESE TWO LINES ARE OUTPUT FROM THE PROGRAM.
1Ø:43:ØØ MONTR
1Ø:43:ØØ MONTR     EXIT                                                          Monitor indicates that execution of
1Ø:43:ØØ MONTR                                                                   your program has ended.
1Ø:43:ØØ MONTR     .
                   %FIN:
1Ø:43:ØØ MONTR     .DEL MYPROG.REL,QAA.CDR,MYPROG.CBL                            Command entered by Batch.
1Ø:43:21 USER      FILES DELETED:
1Ø:43:21 USER      MYPROG.REL                                                    Response to the DELETE  Command.
1Ø:43:21 USER      QAA.CDR
1Ø:43:21 USER      MYPROG.CBL
1Ø:43:21 USER      Ø3 BLOCKS FREED
1Ø:43:21 MONTR
1Ø:43:21 MONTR     .
1Ø:43:22 MONTR     .KJOB DSKB:MYJOB.LOG[1Ø,1164]=/W/Z:4/B/VS:1Ø/VL:2ØØ/VD:D
1Ø:43:25 K-QUE     TOTAL OF 7 BLOCKS IN LPT REQUEST                              This is the LOGOUT dialog, giving
1Ø:43:3Ø KJOB      OTHER JOBS SAME PPN                                           system information.
1Ø:43:43 LGOUT     JOB 24, USER [1Ø,1164]  LOGGED OFF TTY1Ø2   1Ø43 21-MAR-/2
1Ø:43:45 LGOUT     SAVED ALL 4 FILES (25. DISK BLOCKS)
1Ø:43:45 LGOUT     ANOTHER JOB STILL LOGGED IN UNDER [1Ø,1164]
1Ø:43:45 LGOUT     RUNTIME Ø MIN. Ø3.97 SEC

1Ø:43:54 LPMSG     LPTSPL VERSION 4(125)  RUNNING ON LPT3
1Ø:44:2Ø LPMSG     JOB MYJOB FILE DSKB1:MYPROG.LST[1Ø,1164] FOR [1Ø,1164] STARTED  This is more system information.
1Ø:45:2Ø LPMSG     DSKB1:MYPROG.LST[1Ø,1164] DONE
1Ø:45:21 LPMSG     JOB MYJOB FILE DSKB1:MAP.LPT[1Ø,1164] FOR [1Ø,1164] STARTED
1Ø:46:25 LPMSG     DSKB1:MAP.LPT[1Ø,1164] DONE
```

4.4.2 Sample Output from a Job from a Terminal

This example shows the same job described above as it would be entered from a terminal. You would first create the program as a file on disk.

```
IDENTIFICATION DIVISION.
PROGRAM-ID. MYPROG.
ENVIRONMENT DIVISION.
DATA DIVISION.
PROCEDURE DIVISION.
START.
DISPLAY "THIS IS TO SHOW SAMPLE OUTPUT FROM MPB.".
DISPLAY "THESE TWO LINES ARE OUTPUT FROM THE PROGRAM.".
STOP RUN.
```

Then you would make up a control file to compile and execute the COBOL program.

```
.COMPILE MYPROG.CBL
.EXECUTE MYPROG
```

You must then submit the job to Batch using the SUBMIT command.

```
SUBMIT MYJOB
```

When the job is run, the program is compiled and a listing is produced, even though you did not request it. This is because the COBOL compiler always produces a listing. Note that the compiler adds sequence numbers to the listing, even through you did not include these numbers on the program.

```
P R O G R A M   M Y P R O G.                    COBOL 3(43)      22-MAR-72  15:10

0001      IDENTIFICATION DIVISION.
0002      PROGRAM-ID. MYPROG.
0003      ENVIRONMENT DIVISION.
0004      DATA DIVISION.
0005      PROCEDURE DIVISION.
0006      START.
0007      DISPLAY "THIS IS TO SHOW SAMPLE OUTPUT FROM MPB.".
0008      DISPLAY "THESE TWO LINES ARE OUTPUT FROM THE PROGRAM.".
0009      STOP RUN.

NO ERRORS DETECTED
```

Because you did not request it specifically in the EXECUTE command, you will not get a loader map of your program. The log file is printed next as the last of your output. The output from the program is written in the log file because it is output to the terminal and the log file simulates a terminal dialog. The log file also contains some commands that Batch sent to the monitor for you and some additional system information. An annotated log file is shown on the following page. Note that each line in the log file is preceded by the time of day when the line was written. Following the time is a word that describes what kind of information is on each line. You do not have to know what each of these words means because much of the information is system information.

```
15:09:06 BVERS  BATCON 7(36) INP: SUBJOB Ø1 OF Ø6
15:09:06 BDATE  22-MAR-72
15:09:06 BASUM  MPB[1Ø,1164] FOR *SMITH  *[1Ø,1164] LOG FILE IN [1Ø,1164]
                REQUEST CREATED AT  15:Ø7:32 22-MAR-72
                UNIQUE: 2 RESTART: Ø
```

This is sytem information that Batch enters. It need not concern you.

```
15:Ø9:Ø6 MONTR
15:Ø9:Ø6 MONTR  .LOGIN 1Ø,1164
15:Ø9:27 USER   JOB 35     554A1F DUAL CPU TTY1Ø2
15:Ø9:27 USER   OTHER JOBS SAME PPN
15:Ø9:27 USER   15Ø9    22-MAR-72        WED
15:Ø9:51 USER
```

Batch logs your job into the system. The information that follows is the system response.

```
15:1Ø:Ø6 MONTR
15:1Ø:Ø6 MONTR  .SET TIME 3ØØ
15:1Ø:Ø6 MONTR
15:1Ø:Ø6 MONTR  .SET SPOOL ALL
15:1Ø:Ø7 MONTR
15:1Ø:Ø7 MONTR
```

These are commands that Batch enters for you.

```
15:1Ø:Ø7 MONTR  ..COMPILE MYPROG.CBL
15:1Ø:31 USER   COBOL: MYPROG.
15:11:18 MONTR  EXIT
15:11:18        MONTR
```

This is the command from your control file and the response.

```
15:11:18 MONTR  ..EXECUTE MYPROG.CBL
15:11:3Ø USER   LOADING
15:11:3Ø USER
15:11:3Ø USER   COBOL 1K CORE
15:11:3Ø USER   EXECUTION
```

This is another command from your control file and its response.

```
15:11:3Ø USER   THIS IS TO SHOW SAMPLE OUTPUT FROM MPB.
15:11:3Ø USER   THESE TWO LINES ARE OUTPUT FROM THE PROGRAM.
```

This is the output from your program.

```
15:11:3Ø MONTR
15:11:3Ø MONTR  EXIT
15:11:3Ø MONTR
```

This indicates that execution has ended.

```
15:11:3Ø MONTR  .
15:11:3Ø MONTR  .KJOB DSKBØ:MPB.LOG[1Ø,1164]=/W/Z:4/B/VR:1Ø/VS:425/V/L:2ØØ/VD:P
15:11:58 K-QUE  TOTAL OF 4 BLOCKS IN LPT REQUEST
15:15:35 LGOUT  JOB 35. USER [1Ø,1164]  LOGGED OFF TTY1Ø2  1515 22-MAR-72
15:15:36 LGOUT  SAVED ALL 1Ø FILES (125. DISK BLOCKS)
15:15:36 LGOUT  RUNTIME Ø MIN, Ø4.Ø5 SEC
```

This is the LOGOUT dialog, which gives system information.

```
15:15:44 LPMSG  LPTSPL VERSION 4(125)  RUNNING ON LPT3
15:16:Ø6 LPMSG  JOB MPB FILE DSKB1:MYPROG.LST[1Ø,1164] FOR [1Ø,1164] STARTED
15:17:Ø5 LPMSG  DSKB1:MYPROG.LST[1Ø,1164] DONE
```

This is more system information.

4.4.3 Sample Dump

Shown on the following pages is the log file containing an error message and the dump that Batch requested as a result of the message. The error resulted from use of a logical name in a program without assigning the logical name to a physical device at run time.

The dump lists the assembly language equivalent of your program, and the location in memory in octal, decimal, ASCII code, and SIXBIT code. (SIXBIT code is a compressed form of ASCII used in COBOL and some system programs.) Only the first three pages of the dump are shown.

```
14:25:42 BVERS  BATCON 7(36) INP: SUBJOB Ø1 OF Ø6
14:25:42 BDATE  22-MAR-72
14:25:42 BASUM  INJOB[1Ø,1164] FOR *GORFINKLE  *[1Ø,1164] LOG FILE IN
                REQUEST CREATED AT  14:24:35 22-MAR-72
                UNIQUE: 2 RESTART: Ø
14:25:42 MONTR
14:25:42 MONTR  .LOGIN 1Ø,1164
14:25:45 USFR   JOB 3Ø      554A1F DUAL CPU TTY1Ø2
14:25:45 USER   OTHER JOBS SAME PPN
14:25:45 USER   1425    22-MAR-72       WED
14:25:46 MONTR
14:25:46 MONTR  .SET TIME 3ØØ
14:25:46 MONTR
14:25:46 MONTR  .SET SPOOL ALL
14:25:46 MONTR
14:25:46 MONTR  ..COMPILE EXAMPLE.CBL
14:25:47 MONTR
14:25:47 MONTR  EXIT
14:25:47 MONTR
```

```
14:25:47 MONTR  ..EXECUTE EXAMPLE.CBL
14:25:5Ø USER   LOADING
14:25:55 USER
14:25:55 USER   COBOL 1K CORE
14:25:55 USER   EXECUTION
14:25:55 USER
14:25:55 USER   INIT TOOK THE ERROR RETURN
14:25:55 USER    DEVICE MAG1 IS NOT A DEVICE OR IS NOT AVAILABLE TO THIS JOB
14:25:55 USER   INIT TOOK THE ERROR RETURN
14:25:55 USER    DEVICE MAG2 IS NOT A DEVICE OR IS NOT AVAILABE TO THIS JOB
14:25:55 USER   ?LAST COBOL UUO CALLED FROM USER LOCATION 4ØØ1155
14:25:55 MONTR
14:25:55 MONTR  EXIT
14:25:55 MONTR
14:25:55 MONTR  .

14:25:56 MONTR  .CLOSE
14:25:56 MONTR
14:25:56 MONTR
14:25:56 MONTR  .DUMP
14:26:Ø4 USER
14:26:Ø4 USER   Ø SYMBOLS EXTRACTED
14:26:15 MONTR
14:26:15 MONTR  EXIT
14:26:15 MONTR
14:26:15 MONTR  .KJOB DSKB1:INJOB.LOG[1Ø,1164]=/W/Z:4/B/VR:1Ø/VS:422/VL:2ØØ/VD:P
14:26:17 K-QUE  TOTAL OF 3Ø BLOCKS IN LPT REQUEST
14:26:21 KJOB   OTHER JOBS SAME PPN

14:26:31 LGOUT  JOB 3Ø, USER [1Ø,1164]  LOGGED OFF TTY1Ø2  1426 22-MAR-72
14:26:32 LGOUT  SAVED ALL 6 FILES (1ØØ. DISK BLOCKS)
14:26:32 LGOUT  ANOTHER JOB STILL LOGGED IN UNDER [1Ø,1164]
14:26:32 LGOUT  RUNTIME Ø MIN, 1Ø.79 SEC
14:26:4Ø LPMSG  LPTSPL VERSION 4(125)  RUNNING ON LPT3
14:27:Ø4 LPMSG  JOB INJOB FILE DSKBØ:Ø3ØDAE[1Ø,1164] FOR [1Ø,1164] STARTED
14:29:24 LPMSG  DSKBØ:Ø3ØDAE[1Ø,1164] DONE
```

This is the error message that caused Batch to request the dump.

```
QUICK DUMP VERSION %3(24) [FILE SYS:QUIKDM.CCL]

MONITOR INFORMATION

MONITOR NAME    554A1F DUAL CPU        BUILT ON 03-21-72

SYSTEM SERIAL NUMBER IS 160

MONITOR VERSION IS 000000,050400


JOB INFORMATION

DUMP TAKEN 3-22-72 AT 14:25

DAEMON VERSION 6(21)-0

JOB NUMBER 30

TTY102 PPN [10,1164] CHARGE NUMBER 0

RUN TIME =0 MIN. 50 SECONDS

TOTAL KCS =6

TOTAL OF 128 DISK READS, 10 DISK WRITES

PRIV. BITS 0

THERE ARE 0 REAL TIME DEVICES IN USE

CURRENT HPQ IS 0 LAST HPQ COMMAND WAS 0

HISEG NAME   DSKB:LIBOL .SHR

HISEG DIRECTORY [1,4]

USER NAME IS GORFINKLE

USER CORE LIMIT IS 261632 WORDS

USER TIME LIMIT IS 299 SECONDS

PROGRAM NAME IS COBOL
```

```
CORE INFORMATION


PC = 700000,057777 OPC = 000000,000000
 LAST UUO AT 440004,000006

SYMBOLIC LOCATIONS
PC = BLKI   57777
OPC = 7
LAST UUO AT ANDCB  6(4)

ACS IN OCTAL:


0/       010500,000002   000000,000000   522202,715530
3/       554147,220000   000000,000002   000000,000000
6/       000000,000000   200022,001361   000000,001777
11/      000000,000204   777777,000000   000000,001425
14/      310000,001412   004001,001424   000000,000000
17/      777601,001463

ACS IN DECIMAL:


0/       1157627906   0   -23319569576   -19837149184   2   0   0
7/       17184588529   1023   132   -262144   789   26843546378
15/      537133844   0   -33291469

SELECTED CORE AREAS DUMPED AS INSTRUCTION,OCTAL,DECIMAL,SIXBIT,ASCII
```

AROUND C(AC17) [HOPEFULLY A PUSH DOWN LIST]

```
1443/   DPB     5,814        137240,001456   12792628014  +Z@  ,N       J
1444/   UUO002 3,728         002140,001330     293602008   1@  +8       F   L
1445/   UUO012               012000,000000    1342177280  ↓0            @
1446/   JRST    795          254000,001433   23085450011  5@   ,;     +
1447/   PUSHJ   15,815       260740,001457   23748150063  6'@  ,O     ,
1450/   Z       17           000000,000021            17         1
1451/   UUO001 1,728         001040,001330     142607064   (@  +8       "   L
1452/   UUO001 1,661         001040,001225     142606997   (@  *5       "   J
1453/   PUSHJ   15,@113      260760,000161   23752343665  6'P  ↓Q     ,     8
1454/   UUO001 5,753(14)     001256,001361     179831537   *N  +Q      *P   X
1455/   CATL    8,699        301400,001273   25971131067  8,   *[     00    ]
1456.   CATL    8,766        301400,001376   25971131134  8,   +      00
1457/   AOS     (15)         350017,000000   31142445056  * /         ! X
1460/   POPJ    15,          263740,000000   24150802432  6?@         ,
1461/   Z                    000000,000000             0
1462/   Z                    000000,000000             0
1463/   UUO010 131191        010000,400167    1073873015  ↓   @↓W           ;
1464/   CAM     131497       310000,400651   26843677097  9   @&I      2    T
1465/   CAM     131555       310000,400743   26843677155  9   @'C      2    Q
1466/   CAM     132709       310000,403145   26843678309  9   @9E      2    2
1467/   Z                    000000,000000             0
1470/   Z                    000000,000000             0
1471/   Z                    000000,000000             0
1472/   Z                    000000,000000             0
1473/   Z                    000000,000000             0
1474/   Z                    000000,000000             0
1475/   Z                    000000,000000             0
1476/   Z                    000000,000000             0
1477/   Z                    000000,000000             0
1500/   Z                    000000,000000             0
1501/   Z                    000000,000000             0
1502/   Z                    000000,000000             0
1503/   Z                    000000,000000             0
```

CHAPTER 5
PERFORMING COMMON TASKS WITH BATCH

This chapter shows some sample jobs that are run from a terminal and from cards. Section 5.1 illustrates entering jobs from a terminal. Section 5.2 shows entering jobs from cards. The examples are the same in both cases, the difference is only in the way that they are entered.

5.1 USING THE TERMINAL TO ENTER JOBS

ALGOL Example

The first job is a simple ALGOL program that writes output to the terminal. Since the job is being entered through Batch, the output is written in the log file instead of on the terminal.

```
BEGIN
   REAL X; INTEGER I;
   X := 1;
   FOR I := 1 UNTIL 1000 DO X := X+I;
   PRINT(X);
END
```

The control file for the program is as follows.

```
.COMPILE MYPROG.ALG/LIST
.EXECUTE MYPROG.ALG

SUBMIT MYFILE
```

When Batch starts the job, the statements in the control file call the ALGOL compiler to compile the program. Batch then calls the loader to load the program for execution. A listing of the program will be printed with the log file, as shown below.

```
DECSYSTEM 10 ALGOL-60, V, 2A(145):
13-APR-72     15:25:57

000003    B1            1 BEGIN
START OF BLOCK 1
000006                  2          REAL X; INTEGER I;
000006                  3          X :=1;
000016                  4          FOR I :=1 UNTIL 1000 DO X :=X+I;
000023                  5          PRINT(X);
000026          E1      6 END

END BLOCK 1, CONT 0

 0 ERRORS
```

```
15:25:50 BVERS  BATCON 7(52) INP: SUBJOB 01 OF 06
15:25:50 BDATE  13-APR-72
15:25:50 BASUM  MYFILE[10,1461] FOR *SMITH *[10,1461] LOG FILE IN [10,1461]
                REQUEST CREATED AT 15:24:39 13-APR-72
                UNIQUE: 2 RESTART: 0
15:25:50 MONTR
15:25:50 MONTR  .LOGIN 10/1461
15:25:51 USER   JOB 20     55425E DUAL CPU TTY110
15:25:51 USER   OTHER JOBS SAME PPN
15:25:51 USER   1525    13-APR-72       THUR
15:25:52 MONTR
15:25:52 MONTR  .SET TIME 300
15:25:52 MONTR
15:25:52 MONTR  .SET SPOOL ALL
15:25:53 MONTR
15:25:53 MONTR
15:25:53 MONTR  ..COMPILE MYPROG.ALG/LIST
15:25:56 USER   ALGOL: MYPROG
15:25:57 MONTR
15:25:57 MONTR  EXIT
15:25:58 MONTR
15:25:58 MONTR  ..EXECUTE MYPROG.ALG
15:25:58 USER   LOADING
15:26:06 USER
15:26:06 USER   MYPROG 1K CORE
15:26:06 USER   EXECUTION
15:26:07 USER    5.0050100&  5
15:26:07 USER
15:26:07 USER   END OF EXECUTION - 1K CORE
15:26:07 USER
15:26:07 USER   EXECUTION TIME: 0.08 SECS.
15:26:07 USER
15:26:07 USER   ELAPSED TIME:   0.15 SECS.
15:26:07 MONTR
15:26:07 MONTR  .
15:26:07 MONTR  .KJOB DSKB0:MYFILE.LOG[10,1461]=/W/Z:4/B/VR:10/VS:384/VL:200/VD:P
15:26:08 K-QUE  TOTAL OF 3 BLOCKS IN LPT REQUEST
15:26:12 KJOB   OTHER JOBS SAME PPN
15:26:15 LGOUT  JOB 20, USER [10,1461]  LOGGED OFF TTY110    1526  13-APR-72
15:26:15 LGOUT  SAVED ALL 40 FILES (650. DISK BLOCKS)
15:26:15 LGOUT  ANOTHER JOB STILL LOGGED IN UNDER [10,1461]
15:26:15 LGOUT  RUNTIME 0 MIN, 03.25 SEC
15:26:21 LPMSG  LPTSPL VERSION 4A(141)  RUNNING ON LPT2
15:26:42 LPMSG  JOB MYFILE FILE DSKB0:MYPROG.LST[10,1461] FOR [10,1461] STARTED
15:27:35 LPMSG  DSKB0:MYPROG.LST[10,1461] DONE
```

BASIC Example

The next sample shows how to enter a BASIC program to Batch. You must make up the file and save it on disk. Then make up a control file that simulates the dialog with the BASIC system. The program is shown below.

```
5     INPUT D
10    IF D = 2 THEN 110
20    PRINT "X VALUE","SINE","RESOLUTION"
30    FOR X=0 TO 3 STEP D
40    IF SIN(X)<=M THEN 80
50    LETX0=X
60    LET M=SIN(X)
80    NEXT X
90    PRINT X0, M,D
100   GO TO 5
110   END
```

The program requests data from the user when it is running. You include the data in the control file. The final data item must be 2 to conclude the program. The control file follows.

```
.R BASIC
*OLD
*DSK:MYBAS.BAS
*RUN
.1
.01
.001
2
*BYE
```

The output from the program will be printed in the control file because it would normally be printed on the terminal. The command to submit the job to Batch is as follows.

```
SUBMIT = BAS.CTL
```

```
15:41:37 BVERS   BATCON 7(52) INP: SUBJOB 02 OF 06
15:41:37 BDATE   13-APR-72
15:41:37 BASUM   BAS[10,1461] FOR *SMITH *[10,1461] LOG FILE IN [10,1461]
                 REQUEST CREATED AT  15:40:23 13-APR-72
                 UNIQUE: 2 RESTART: 0
15:41:37 MONTR
15:41:37 MONTR   .LOGIN 10/1461
15:41:39 USER    JOB 15     55425E DUAL CPU TTY115
15:41:40 USER    OTHER JOBS SAME PPN
15:41:40 USER    1541     13-APR-72        THUR
15:41:41 MONTR
15:41:41 MONTR   .SET TIME 300
15:41:41 MONTR
15:41:41 MONTR   .SET SPOOL ALL
15:41:41 MONTR
15:41:41 MONTR
15:41:41 MONTR   ..R BASIC
15:41:41 USER
15:41:42 USER
15:41:42 USER    NEW OR OLD--*OLD
```

```
15:41:42 USER   OLD FILE NAME--*DSK:MYBAS
15:41:43 USER
15:41:43 USER   READY
15:41:43 USER   *RUN
15:41:47 USER
15:41:47 USER   MYBAS             15:41           13-APR-72
15:41:47 USER
15:41:47 USER
15:41:47 USER
15;41:47 USER    ?.1
15:41:47 USER   X VALUE           SINE            RESOLUTION
15:41:47 USER    3.                0.14112         0.1
15:41:47 USER    ?.01
15:41:47 USER   X VALUE           SINE            RESOLUTION
15:41:48 USER    3.                0.141121        0.01
15:41:48 USER    ?.001
15:41:48 USER   X VALUE           SINE            RESOLUTION
15:41:48 USER    2.99999           0.14113         0.001
15:41:49 USER    ?2
15:41:49 USER
15:41:49 USER
15:41:49 USER   TIME:  1.50 SECS.
15:41:49 USER
15:41:50 USER   READY
15:41:50 USER   *BYE
15:41:50 USER   JOB 15, USER [10,1461] LOGGED OFF TTY115   1541  13-APR-72
15:41:52 USER   SAVED ALL 33 FILES (600. DISK BLOCKS)
15:41:52 USER   ANOTHER JOB STILL LOGGED IN UNDER [10,1461]
15:41:53 MONTR  RUNTIME 0 MIN, 03.05 SEC
```

FORTRAN Example

The third example shows a FORTRAN program that prints output on the line printer. In the control file, you want to tell Batch to delete your relocatable binary file if an error occurs when your program is executed. Otherwise, you want Batch to save your relocatable binary file as it normally would. The program is shown below.

```
C       THIS PROGRAM CALCULATES PRIME NUMBERS FROM 11 TO 50.
        DO 10 I =11,50,2
        J=1
4       J=J+2
        A=J
        A=I/A
        L=I/J
        B=A-L
        IF (B) 5,10,5
5       IF (J.LT.SQRT(FLOAT(I))) GO TO 4
        PRINT 105,I
10      CONTINUE
105     FORMAT (I4, ' IS PRIME.')
        END
```

The control file to compile and execute this program, deleting the relocatable binary file if there is an execution error, is as follows.

```
.COMPILE MYPROG.F4
.EXECUTE MYPROG.F4
.IF (NOERROR) .GOTO END
.DELETE MYPROG.REL
END:: ;END OF JOB
```

The command to submit this job is as follows.

```
SUBMIT MYFOR.CTL,MYFOR.LOG/DISPOSE:DELETE
```

The log file will be deleted after the output has been printed.

```
09:50:07 BVERS  BATCON 7(52) INP: SUBJOB 02 OF 06
09:50:07 BDATE  14-APR-72
09:50:07 BASUM  MYFOR[10,1461] FOR *SMITH :[10,1461] LOG FILE IN [10,1461]
                REQUEST CREATED AT 09:49:19 14-APR-72
                UNIQUE: 2 RESTART: 0
09:50:07 MONTR
09:50:07 MONTR  .LOGIN 10/1461
09:50:09 USER   JOB 23     55425I DUAL CPU TTY115
09:50:09 USER   OTHER JOBS SAME PPN
09:50:13 USER   0950    14-APR-72       FRI
09:50:13 MONTR
09:50:13 MONTR  .SET TIME 300
09:50:14 MONTR
09:50:14 MONTR  .SET SPOOL ALL
09:50:14 MONTR
09:50:14 MONTR  ..COMPILE MYPROG.F4
09:50:16 USER   FORTRAN:  MYPROG.F4
09:50:17 MONTR
09:50:17 MONTR  EXIT
09:50:17 MONTR
09:50:17 MONTR  ..EXECUTE MYPROG.F4
09:50:17 USER   LOADING
09:50:23 USER
09:50:23 USER   MYPROG 2K CORE
09:50:23 USER   EXECUTION
09:50:23 USER
09:50:23 USER   11 IS PRIME.
09:50:23 USER   13 IS PRIME.
09:50:23 USER   17 IS PRIME.
09:50:23 USER   19 IS PRIME.
09:50:23 USER   23 IS PRIME.
09:50:23 USER   29 IS PRIME.
09:50:23 USER   31 IS PRIME.
09:50:23 USER   37 IS PRIME.
09:50:23 USER   41 IS PRIME.
09:50:23 USER   43 IS PRIME
09:50:23 USER   47 IS PRIME.
09:50:23 USER
09:50:23 USER   CPU TIME: 0.37  ELAPSED TIME: 0.60
09:50:23 USER   NO EXECUTION ERRORS DETECTED
09:50:25 MONTR
09:50:25 MONTR  EXIT
09:50:25 MONTR
09:50:27 MONTR
09:50:27 MONTR
                .
                END:
                 ;END OF JOB
```

```
09:50:27 MONTR  .KJOB DSKB1:MYFOR.LOG[10,1461]=/W/Z:4/B/VR:10/VS:420/VL:200/VD:P
09:50:28 K-QUE  TOTAL OF 3 BLOCKS IN LPT REQUEST
09:50:32 KJOB   OTHER JOBS SAME PPN
09:50:34 LGOUT  JOB 23, USER [10,1461] LOGGED OFF TTY115   0950  14-APR-72
09:50:34 LGOUT  SAVED ALL 33 FILES (610. DISK BLOCKS)
09:50:34 LGOUT  ANOTHER JOB STILL LOGGED IN UNDER [10,1461]
09:50:34 LGOUT  RUNTIME 0 MIN, 05.39 SEC
```

COBOL Example

The fourth program shows a COBOL program that reads a magnetic tape and writes output on another magnetic tape. To have your magnetic tapes mounted on drives and assigned to you, you must request that the operator mount them. Since you do not know which drives will be assigned to your job, you must assign them in your job with logical device names. The MOUNT command assigns the drive to your job and associates the logical name that you specify in it with the physical drive assigned. You should include a PLEASE command to the operator to tell him that you want two magnetic tape drives. If he can't let you have the drives because they are in use, you can ask him to enter your job again. Your magnetic tapes, one with the input data, the other blank so that you can write on it, should be given to the operator or kept at the central site, so that the operator can find your tapes. The program is as follows.

```
IDENTIFICATION DIVISION.
ENVIRONMENT DIVISION.
INPUT-OUTPUT SECTION.
FILE-CONTROL.
     SELECT INFIL, ASSIGN MAG1.
     SELECT OUTFIL, ASSIGN MAG2.
DATA DIVISION.
FILE SECTION.
FD   INFIL, LABEL RECORDS ARE STANDARD,
     VALUE OF IDENTIFICATION IS "INFIL DAT",
     BLOCK CONTAINS 20 RECORDS.
01   INREC, PIC X(80).
FD   OUTFIL, LABEL RECORDS ARE STANDARD,
     VALUE OF IDENTIFICATION IS "OUTFILDAT",
     BLOCK CONTAINS 12 RECORDS.
01   OUTREC, PIC X(80).
PROCEDURE DIVISION.
START.
     OPEN INPUT INFIL, OUTPUT OUTFIL.
LOOP.
     READ INFIL; AT END GO TO FIN.
     WRITE OUTREC FROM INREC.
     GO TO LOOP.
FIN.
     CLOSE OUTFIL, INFIL.
     STOP RUN.
```

The control file and the SUBMIT command to enter this program to Batch is as follows.

```
.PLEASE NEED TWO MAGTAPES, IF I CAN'T HAVE THEM,  REQUEUE.
.MOUNT MTA:MAG1/VID:INFIL /RONLY
.MOUNT MTA:MAG2/VID:OUTFIL/WENABLE
.COMPILE MYPROG.CBL
.EXECUTE MYPROG.CBL
.DISMOUNT MAG1:
.DISMOUNT MAG2:
.DELETE MYPROG.*

.SUBMIT MYJOB=MYJOB.CTL
```

The log file is shown below.

```
11:53:26 BVERS  BATCON 7(53) INP: SUBJOB 01 OF 06
11:53:26 BDATE  20-APR-72
11:53:26 BASUM  MYJOB[10,1416] FOR *SMITH *[10,1461] LOG FILE IN [10,1461]
                REQUEST CREATED AT 11:52:31 20-APR-72
                UNIQUE: 2 RESTART: 0
11:53:26 MONTR
11:53:26 MONTR  .LOGIN 10/1461
11:53:30 USER   JOB 17     554250 DUAL CPU TTY103
11:53:30 USER   OTHER JOBS SAME PPN
11:53:30 USER   1153    20-APR-72       THURS
11:53:30 MONTR
11:53:30 MONTR  .SET TIME 300
11:53:30 MONTR
11:53:30 MONTR  .SET SPOOL ALL
11:53:30 MONTR
11:53:30 MONTR  ..PLEASE NEED TWO MAG TAPES, IF I CAN'T HAVE THEM, REQUEUE.
11:53:50 MONTR  .MOUNT MTA:MAG1/VID:INFIL/RONLY
11:53:50 USER   OPERATOR NOTIFIED
11:53:52 USER   WAITING...
11:54:19 USER   MAG1 (MTA1)  MOUNTED
11:54:19 USER
11:54:21 MONTR  ..MOUNT MTA:MAG2/VID:OUTFIL/WENABL
11:54:22 USER   OPERATOR NOTIFIED
11:54:25 USER   WAITING...
11:57:23 USER   MAG2 (MTA2)  MOUNTED
11:57:23 USER
11:57:23 MONTR  ..COMPILE MYPROG.CBL
11:57:25 MONTR
11:57:25 MONTR  EXIT
11:57:25 MONTR
11:57:25 MONTR  ..EXECUTE MYPROG.CBL
11:57:48 USER   LOADING
11:58:05 USER
11:58:05 USER   COBOL 1K CORE
11:58:05 USER   EXECUTION
11:58:05 MONTR
11:58:05 MONTR  EXIT
11:58:05 MONTR
11:58:05 MONTR  ..DISMOUNT MAG1:
11:58:12 USER   OPERATOR NOTIFIED
11:58:12 USER   WAITING...
11:58:45 USER   MAG1 DISMOUNTED
11:58:46 MONTR
11:58:46 MONTR  ..DISMOUNT MAG2:
11:58:58 USER   OPERATOR NOTIFIED
11:58:58 USER   WAITING...
12:00:07 USER   MAG2 DISMOUNTED
```

```
12:00:07 MONTR
12:00:07 MONTR   .
12:00:07 MONTR   .KJOB DSKB0:MYJOB.LOG[10,1461]=/W/Z:4/B/VR:10/VS:607/VL:200/VP:10/VD
12:00:10 K-QUE   TOTAL OF 4 BLOCKS IN LPT REQUEST
12:00:14 KJOB    OTHER JOBS SAME PPN
12:00:17 LGOUT   JOB 17, USER [10,1461] LOGGED OFF TTY103    1200  20-APR-72
12:00:18 LGOUT   SAVED ALL 38 FILES (645. DISK BLOCKS)
12:00:18 LGOUT   ANOTHER JOB STILL LOGGED IN UNDER [10,1461]
12:00:18 LGOUT   RUNTIME 0 MIN, 06.39 SEC
12:22:10 LPMSG   LPTSPL VERSION 4A(141)  RUNNING ON LPT0
```

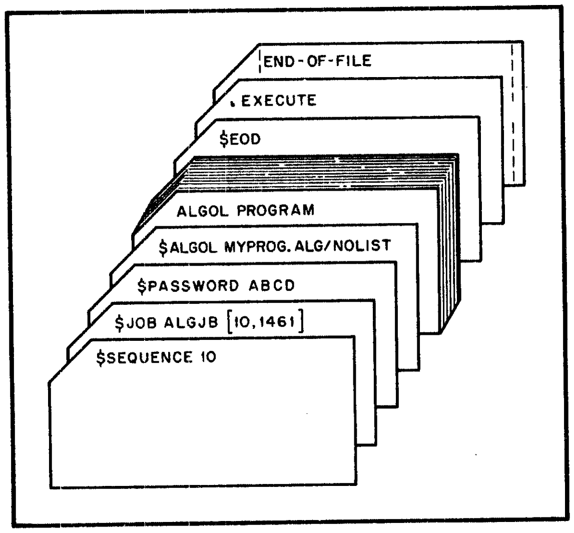
5.2 USING CARDS TO ENTER JOBS

ALGOL Example

The first job is a simple ALGOL program that writes its output into the log file because it has statements that would cause it normally to write to the terminal. The program is as follows.

```
BEGIN
      REAL X;  INTEGER I;
      X := 1;
      FOR I := 1 UNTIL 1000 DO X := X+I;
      PRINT(X);
END
```

The cards to enter this program are as follows.



10-0925

The control file that MPB makes up for you contains the following commands.

```
.COMPILE MYPROG.ALG /COMPILE /LIST
.EXECUTE
```

The output, including the log file is shown below.

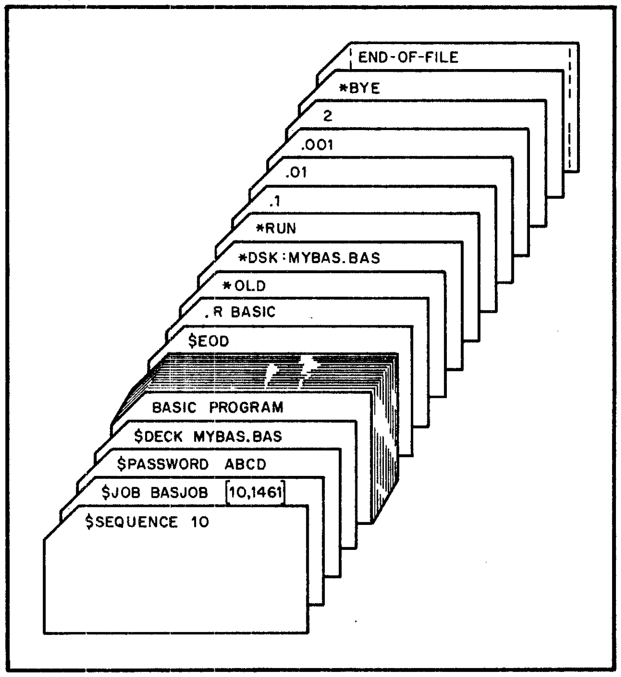
```
09:01:43 DATE    13-APR-72    55425E DUAL CPU   CDRSTK VER 12(26) DSK
09:01:43 CARD    $JOB ALGJB[10/1461]
                 $ALGOL MYPROG.ALG/NOLIST

09:01:45 STSUM   END OF FILE AFTER 12 CARDS, 03 FILES, 03 BLKS

09:01:53 BVERS   BATCON 7(52) INP: SUBJOB 02 OF 06
09:01:53 BDATE   13-APR-72
09:01:53 BASUM   ALGJB[10,1461] FOR **[10,1461] LOG FILE IN [10,1461]
                 REQUEST CREATED AT 09:01:08 13-APR-72
                 UNIQUE: 2 RESTART: 1

09:01:53 MONTR
09:01:53 MONTR   .LOGIN 10.1461
09:01:58 USER    JOB 13      55425E DUAL CPU TTY115
09:01:58 USER    OTHER JOBS SAME PPN
09:01:58 USER    0901    13-APR-72       THUR
09:01:58 MONTR
09:01:58 MONTR   .SET TIME 300
09:01:58 MONTR
09:01:58 MONTR   .SET SPOOL ALL
09:01:59 MONTR
09:01:59 MONTR   .
                 $JOB ALGJB[10/1461]
09:02:01 MONTR   .
                 $ALGOL MYPROG.ALG/NOLIST
09:02:01 MONTR   .COMP /COMPILE MYPROG.ALG /N    ;CREATED BY CDRSTK
09:02:08 USER    ALGOL: MYPROG
09:02:08 MONTR
09:02:08 MONTR   EXIT
09:02:08 MONTR
09:02:08 MONTR   .
                 $EOD
09:02:08 MONTR   .EXECUTE
09:02:10 USER    ALGOL: MYPROG
09:02:27 USER    LOADING
09:02:39 USER
09:02:39 USER    MYPROG 1K CORE
09:02:39 USER    EXECUTION
09:02:41 USER     5.0050100&  5
09:02:41 USER
09:02:41 USER    END OF EXECUTION  - 1K CORE
09:02:41 USER
09:02:41 USER    EXECUTION TIME: 0.08 SECS.
09:02:41 USER
09:02:41 USER    ELAPSED TIME:   0.12 SECS.
09:02:41 MONTR   .
                 %FIN:
09:02:41 MONTR   .DEL MYPROG.REL,MYPROG.ALG
09:02:42 USER    FILES DELTED:
09:02:42 USER    MYPROG.REL
09:02:43 USER    MYPROG.ALG
09:02:43 USER    02 BLOCKS FREED
09:02:43 MONTR
09:02:43 MONTR   .
09:02:44 MONTR   .KJOB DSKB:ALGJB.LOG[10,1461]=/W/Z:4/B/VS:320/VL:10/VD:D
09:02:45 K-QUE   TOTAL OF 4 BLOCKS IN LPT REQUEST
```

BASIC Example

The next example shows entering a BASIC program. You must include the program after a $DECK card so that it will be copied into a file on disk. No $DATA card can be used because BASIC does not use the EXECUTE command and because the data is entered in the control file. The program requests data when it is running; it finds the data in the control file. The final data item must be 2 so that the program can be concluded. The program is shown below.

```
5     INPUT D
10    IF D = 2 THEN 110
20    PRINT "X VALUE", "SINE", "RESOLUTION"
30    FOR X = 0 TO 3 STEP D
40    IF SIN(X) =M THEM 80
50    LET X0 = X
60    LET M = SIN(X)
80    NEXT X
90    PRINT X0, M, D
100   GO TO 5
110   END
```

The cards to enter the program and run it are as follows.



10-0926

The output from the program will be printed in the log file because it would normally be printed on the terminal. The log file is shown below.

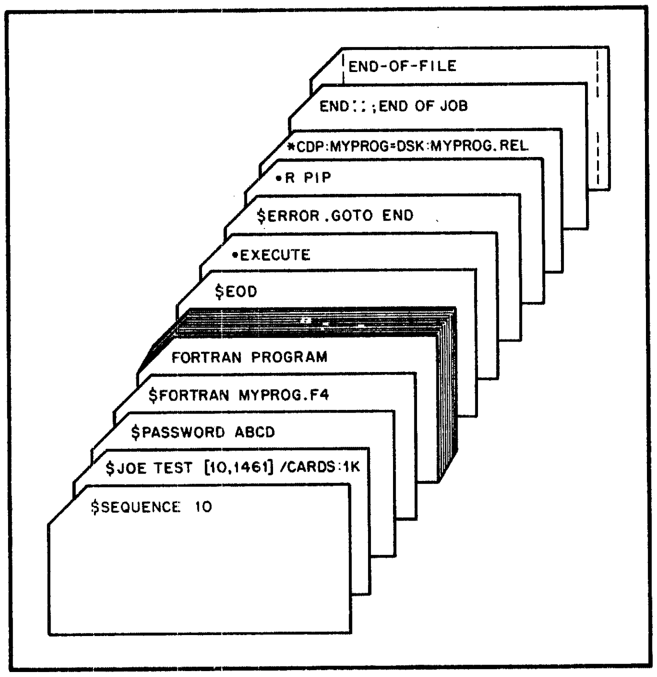
```
11:10:45 DATE   13-APR-72  55425E DUAL CPU CDRSTK VER 12 (26) DSK
11:10:45 CARD   $JOB BASJOB[10/1461]
11:10:46 STSUM  END OF FILE AFTER 24 CARDS, 03 FILES, 04 BLKS

11:10:49 BVERS  BATCON 7(52) INP: SUBJOB 01 OF 14
11:10:49 BDATE  13-APR-72
11:10:49 BASUM  BASJOB[10,1461] FOR **[10,1461] LOG FILE IN [10,1461]
                REQUEST CREATED AT 11:09:57 13-APR-72
                UNIQUE: 2 RESTART: 1
11:10:49 MONTR
11:10:49 MONTR  .LOGIN 10,1461
11:10:51 USER   JOB 19     55425E DUAL CPU TTY114
11:10:51 USER   OTHER JOBS SAME PPN
11:10:52 USER   1110    13-APR-72       THUR
11:10:53 USER
11:10:53 MONTR
11:10:53 MONTR  .SET TIME 300
11:10:53 MONTR
11:10:53 MONTR  .SET SPOOL ALL
11:10:53 MONTR
11:10:53 MONTR  .
                $JOB BASJOB[10/1461]
                $DECK MYBAS.BAS
                $EOD
11:10:53 MONTR  .R BASIC
11:10:53 USER
11:10:54 USER
11:10:54 USER   NEW OR OLD--*OLD
11:10:55 USER   OLD FILE NAME--*DSK:MYBAS
11:10:55 USER
11:10:55 USER   READY
11:10:56 USER   *RUN
11:10:56 USER
11:10:56 USER   MYBAS            11:10           13-APR-72
11:10:56 USER
11:10:56 USER
11:10:56 USER
11:10:56 USER    ?.1
11:10:57 USER   X VALUE          SINE            RESOLUTION
11:10:57 USER    3.               0.14112         0.1
11:10:57 USER    ?.01
11:10:59 USER   X VALUE          SINE            RESOLUTION
11:10:59 USER    3.               0.141121        0.01
11:10:59 USER    ?.001
11:10:59 USER   X VALUE          SINE            RESOLUTION
11:11:00 USER    2.99999          0.14113         0.001
11:11:00 USER    ?2
11:11:00 USER
11:11:00 USER
11:11:00 USER   TIME:  1.50 SECS.
11:11:00 USER
11:11:01 USER   READY
11:11:01 USER   *BYE
11:11:03 USER   JOB 19, USER [10,1461] LOGGED OFF TTY114   1111  13-APR-72
11:11:03 USER   SAVED ALL 33 FILED (600, DISK BLOCKS)
11:11:03 USER   ANOTHER JOB STILL LOGGED IN UNDER [10,1461]
11:11:03 MONTR  RUNTIME 0 MIN, 03.05 SEC
```

FORTRAN Example

The third example shows a FORTRAN program that prints output on the line printer. In the control file, you want to tell Batch to punch your relocatable binary program if it executes correctly. Otherwise, you want to end your job so that you can find your error from the message in the log file. The program is shown below.

```
C       THIS PROGRAM CALCULATES PRIME NUMBERS FROM 11 TO 50.
        DO 10 I=11,50,2
        J=1
4       J=J+2
        A=J
        A=I/A
        L=I/J
        B=A-L
        IF (B) 5,10,5
5       IF (J.LT.SQRT(FLOAT(I))) GO TO 4
        PRINT 105,I
10      CONTINUE
105     FORMAT (I4, ' IS PRIME. ')
        END
```

The cards used to enter this program are as follows.



10-0927

Batch puts the following commands into the control file as a result of the cards you entered.

```
.COMPILE MYPROG.F4 /COMPILE /LIST
.EXECUTE MYPROG.REL /MAP:MAP.LPT
.IF (ERROR) .GOTO END
.R PIP
*CDP:MYPROG = DSK:MYPROG.REL
END: :  ;END OF JOB
```

The printed output from the job, including the log file is shown below.

```
\MYPROG,F4        F40      V25      12-APR-72        13:43   PAGE 1


                          DO 10 I=11,50,2
                          J=1
                   4      J=J+2
                          A=J
                          A=I/A
                          L=I/J
                          B=A-L
                          IF (B) 5,10,5
                   5      IF (J.-T.SQRT (FLOAT (I))) GO TO 4
                          TYPE 105,I
                   10     CONTINJE
                   105    FORMAT (I4, '#IS PRIME.')
                          END
SUBPROGRAMS

FORSE.  JOBFF   FLOAT   SQRT   INTO.   INTI.   EXIT

SCALARS

I        61             J      62     A       63     L       64   B    65



13:43:01 DATE    12-APR-72       554A4B DUAL CPU     CDRSTK VER 12(26) DSK
13:43:01 CARD    $JOB TEST[10,1461]/CARD:1K
13:43:03 STSUM   END OF FILE AFTER 19 CARDS, 03 FILES, 04 BLKS

13:43:21 BVERS   BATCON 7(52) INP: SUBJOB 01 OF 14
13:43:21 BDATE   12-APR-72
13:43:21 BASUM   TEST[10,1461] FOR **[10,1461] LOG FILE IN [10,1461]
                 REQUEST CREATED AT  13:42:04 12-APR-72
                 UNIQUE: 2 RESTART: 1

13:43:21 MONTR
13:43:21 MONTR   .LOGIN 10,1461
13:43:24 USER    JOB 11       554A4B DUAL CPU TTY102
13:43:24 USER    OTHER JOBS SAME PPN
13:43:26 USER    1343    12-APR-72       WED
13:43:28 MONTR
13:43:28 MONTR   .SET TIME 20
13:43:28 MONTR
13:43:28 MONTR   .SET SPOOL A_L
13:43:28 MONTR
13:43:28 MONTR   .
                 $JOB TEST[10,1461]/CARD:1K
                 $FORTRAN MYPROG.F4
```

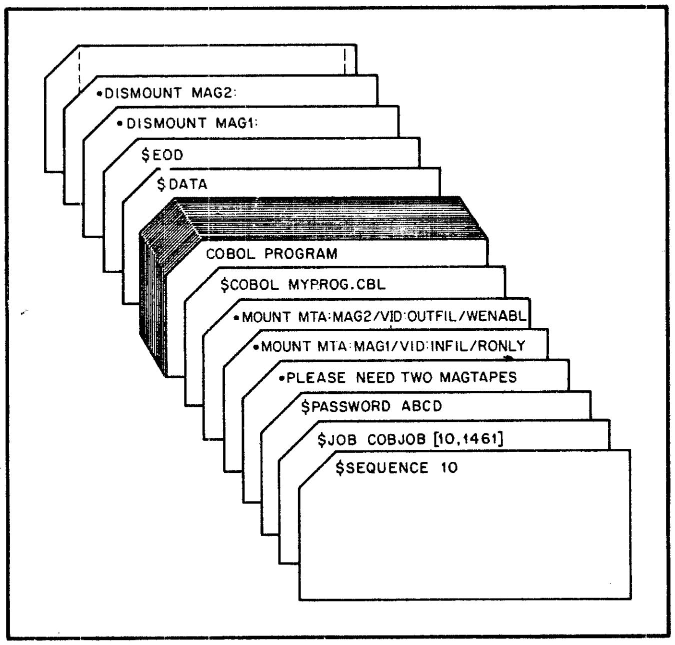
```
13:43:28 MONTR  .COMP /COMPILE MYPROG.F4/LIST     ;CREATED BY CDRSTK
13:43:30 USER   FORTRAN:  MYPROG.F4
13:43:33 MONTR
13:43:33 MONTR  EXIT
13:43:33 MONTR
13:43:33 MONTR  .
                $EOD
13:43:33 MONTR  .EXECUTE
13:43:34 USER   FORTRAN:  MYPROG.F4
13:43:37 USER   LOADING
13:43:41 USER
13:43:41 USER   MYPROG 2K CORE
13:43:42 USER   EXECUTION
13:43:42 USER
13:43:42 USER   11 IS PRIME.
13:43:42 USER   13 IS PRIME.
13:43:42 USER   17 IS PRIME.
13:43:42 USER   19 IS PRIME.
13:43:42 USER   23 IS PRIME.
13:43:42 USER   29 IS PRIME.
13:43:42 USER   31 IS PRIME.
13:43:43 USER   37 IS PRIME.
13:43:43 USER   41 IS PRIME.
13:43:43 USER   43 IS PRIME.
13:43:43 USER   47 IS PRIME.
13:43:43 USER
13:43:43 USER   CPU TIME: 0.27  ELAPSED TIME: 1.82


13:43:43 USER   NO EXECUTION ERRORS DETECTED
13:43:43 MONTR
13:43:43 MONTR
13:43:43 MONTR  EXIT
13:43:43 MONTR  .R PIP
13:43:43 USER   *CDP: MYPROG=DSK:MYPROG.REL
13:43:43 MONTR  .
                END:
                ;END OF JOB
13:43:44 MONTR  .
                %FIN:
13:43:44 USER   .DEL MYPROG.REL,MYPROG.F4
13:43:44 USER   FILES DELETED:
13:43:45 USER   MYPROG.REL
13:43:46 USER   MYPROG.F4
13:43:46 USER   03 BLOCKS FREED
13:43:46 MONTR
13:43:46 MONTR  .
13:43:48 MONTR  .KJOB DSKB:TEST.LOG[10,1461]=/W/Z:4/P/VS:277/VL:200/VD:D
13:43:48 K-QUE  TOTAL OF 6 BLOCKS IN LPT REQUEST
13:43:52 KJOB   OTHER JOBS SAME PPN
13:43:54 LGOUT  JOB 11, USER [10,1461] LOGGED OFF TTY102   1343  12-APR-72
13:43:54 LGOUT  SAVED ALL 30 FILES (585, DISK BLOCKS)
13:43:54 LGOUT  ANOTHER JOB STILL LOGGED IN UNDER [10,1461]
13:43:54 LGOUT  RUNTIME 0 MIN, 05,64 DEC
13:43:57 LPMSG  LPTSPL VERSION 4A(141)  RUNNING ON LPT1
13:44:02 LPMSG  JOB TEST FILE DSKB1:MYPROG.LST[10,1461] FOR [10,1461] STARTED
13:44:09 LPMSG  DSKB1:MYPROG.LST[10,1461] DONE
```

COBOL Example

The fourth program shows a COBOL program that reads data from a magentic tape and writes output on another magnetic tape. To have your magnetic tapes mounted on drives and assigned to you, you must request that the operator mount them. Since you do not know which drives will be assigned to your job, you must assign them in your job with logical device names. The MOUNT command assigns the drive to your job and associates the logical name that you specify in it with the physical drive assigned. You should include a PLEASE command to the operator to tell him that you want two magnetic tape drives. If he can't let you have the drives because they are in use, you can ask him to enter your job again. Your magnetic tapes, one with the input data, the other blank so that you can write on it, should be given to the operator with your card deck or kept at the central site, so that the operator can find your tapes. The program is as follows.

```
IDENTIFICATION DIVISION.
ENVIRONMENT DIVISION.
INPUT-OUTPUT SECTION.
FILE-CONTROL.
       SELECT INFIL, ASSIGN MAG1.
       SELECT OUTFIL, ASSIGN MAG2.
DATA DIVISION.
FILE SECTION.
FD     INFIL, LABEL RECORDS ARE STANDARD,
       VALUE OF IDENTIFICATION IS "INFIL DAT",
       BLOCK CONTAINS 20 RECORDS.
01     INREC, PIC X(80).
FD     OUTFIL, LABEL RECORDS ARE STANDARD,
       VALUE OF IDENTIFICATION IS "OUTFILDAT",
       BLOCK CONTAINS 12 RECORDS.
01     OUTREC, PIC X(80).
PROCEDURE DIVISION.
START.
       OPEN INPUT INFIL, OUTPUT OUTFIL.
LOOP.
       READ INFIL; AT END GO TO FIN.
       WRITE OUTREC FROM INREC.
       GO TO LOOP.
FIN.
       CLOSE OUTFIL, INFIL.
       STOP RUN.
```

The cards to enter this job are shown below.



10-0928

Batch puts the following commands into the control file for you.

```
.PLEASE NEED TWO MAG TAPES, IF I CAN'T HAVE THEM, REQUEUE.
.MOUNT MTA:MAG1/VID:INFIL /RONLY
.MOUNT MTA:MAG2/VID:OUTFIL /WENABL
.COMPILE /COMPILE MYPROG.CBL /LIST
.EXECUTE MYPROG.REL /MAP:MAP.LPT
.DISMOUNT MAG1:
.DISMOUNT MAG2:
```

The printed output from your job is shown below.

```
ØØ1462 IS THE LOW  SEGMENT BREAK

MAP        STORAGE MAP     15:42 2Ø-APR-72

STARTING ADDRESS   ØØ14Ø6   PROG COBOL    FILE MYPROG

.COMM.    ØØØ14Ø   ØØ1Ø4Ø

COBOL      ØØ12ØØ    ØØ1317 FILES. ØØØ14Ø USES.  ØØØ141  SEGWD.  ØØØ142
         START.    ØØ14Ø6   OVRFN. ØØØ144 POINT. ØØØ145  COMMA.  ØØØ146
         ALTER.    ØØØ143   MEMRY. ØØØ152 TRAC1. ØØØ152  TRAC2.  ØØØ153
         MONEY.    ØØØ147   %NM.   ØØØ155 %DT.   ØØØ156  %PR.    ØØØ157
         TRAC3.    ØØØ154

TRACED     ØØ1457    ØØØØØ3 PTFLG. ØØ1461 TRACE. ØØ1457  TRPD.   ØØ146Ø
         BTRAC.    ØØ146Ø
         TRPOP.    ØØ146Ø

COBOL 1K CORE, 2Ø5 WORDS FREE
LOADER USED 2+4K CORE
```

```
P R O G R A M    C O B O L .                    COBOL 3(43)      2Ø-APR-72

15:41            PAGE 1

         IDENTIFICATION DIVISION.
         ENVIRONMENT DIVISION.
         INPUT-OUTPUT SECTION.
         FILE-CONTROL.
             SELECT INFIL, ASSIGN MAG1.
             SELECT OUTFIL, ASSIGN MAG2.
         DATA DIVISION.
         FILE SECTION.
         FD  INFIL, LABEL RECORDS ARE STANDARD,
              VALUE OF IDENTIFICATION IS "INFIL DAT",
              BLOCK CONTAINS 2Ø RECORDS.
         Ø1   INREC, PIC X(8Ø).
         FD   OUTFIL, LABEL RECORDS ARE STANDARD,
              VALUE OF IDENTIFICATION IS "OUTFILDAT".
              BLOCK CONTAINS 12 RECORDS.
         Ø1   OUTREC, PIC  X(8Ø).
         PROCEDURE DIVISION.
         START.
             OPEN INPUT INFIL, OUTPUT OUTFIL.

         LOOP
             READ INFIL; AT END GO TO FIN.
             WRITE OUTREC FROM INREC.
             GO TO LOOP.
         FIN.
             CLOSE OUTFIL, INFIL.
             STOP RUN.

NO ERRORS DETECTED


15:37:37 DATE    2Ø-APR-72      55425Ø DUAL CPU   DSRSTK VER 12(26) DSK
15:37:37 CARD    $JOB COBJOB[1Ø/1461]
15:37:38 STSUM   END OF FILE AFTER 37 CARDS, Ø4 FILES, Ø6 BLKS

15:37:46 BVERS   BATCON 7(53) INP: SUBJOB Ø1 OF Ø6
15:37:46 BDATE   2Ø-APR-72
15:37:46 BASUM   COBJOB[1Ø,1461] FOR **[1Ø,1461] LOG FILE IN [1Ø,1461]
                 REQUEST CREATED AT 15:36:34 2Ø-APR-72
                 UNIQUE: 2 RESTART: 1
15:37:46 MONTR
15:37:46 MONTR   .LOGIN 1Ø/1461
15:37:48 USER    JOB 24      554250 DUAL CPU TTY1Ø3
15:37:5Ø USER    OTHER JOBS SAME PPN
15:37:53 USER    1537    2Ø-APR-72        THUR
15:37:53 MONTR
15:37:53 MONTR   .SET TIME 3ØØ
15:37:54 MONTR
15:37:54 MONTR   .SET SPOOL ALL
15:37:54 MONTR
15:37:54 MONTR   .
                 $JOB COBJOB[1Ø/1461]
15:37:54 MONTR   .PLEASE NEED TWO MAG TAPES. IF CAN'T HAVE THEM, REQUEUE.
15:38:17 MONTR   .MOUNT MTA:MAG1/VID:INFIL/RONOLY
15:38:18 USER    OPERATOR NOTIFIED
15:38:18 USER    WAITING...
15:39:59 USER    MAG1 (MTAØ)  MOUNTED
15:39:59 USER
```

```
15:39:59 MONTR   ..MOUNT MTA:MAG2/VID:OUTFIL/WENABL
15:40:01 USER    OPERATOR NOTIFIED
15:40:01 USER    WAITING...
15:41:31 USER    MAG2 (MTA1)  MOUNTED
15:41:31 USER
15:41:31 MONTR   .
                 $COBOL MYPROG.CBL
15:41:31 MONTR   .COMP /COMPILE MYPROG.CBL/LIST   ;CREATED BY CDRSTK
15:41:35 USER    COBOL:
15:41:59 MONTR   EXIT
15:41:59 MONTR
15:41:59 MONTR   .
                 $DATA
15:41:59 MONTR   .SET CDR QAA.CDR                   ;CREATED BY CDRSTK
15:41:59 MONTR
15:41:59 MONTR   ..EXEC./MAP:MAP.LPT /REL MYPROG.REL    ;CREATED BY CDRSTK
15:42:04 USER    LOADING
15:42:04 USER    001462 IS THE LOW  SEGMENT BREAK
15:42:05 USER
15:42:06 USER    COBOL 1K CORE
15:42:07 USER    EXECUTION
15:42:09 MONTR
15:42:09 MONTR   EXIT
15:42:09 MONTR
15:42:09 MONTR   .
                 $EOD
15:42:09 MONTR   .DISMOUNT MAG1:
15:42:09 USER    OPERATOR NOTIFIED
15:42:10 USER    WAITING...
15:42:29 USER    MAG1 DISMOUNTED
15:42:29 MONTR
15:42:29 MONTR   ..DISMOUNT MAG2:
15:42:30 USER    OPERATOR NOTIFIED
15:42:31 USER    WAITING...
15:42:47 USER    MAG2 DISMOUNTED
15:42:47 MONTR
15:42:47 MONTR   .
                 %FIN:
15:42:47 MONTR   .DEL MYPROG.REL,QAA.CDR,MYPROG.CBL
15:42:51 USER    FILES DELETED:
15:42:53 USER    MYPROG.REL
15:42:57 USER    QAA.CDR
15:42:59 USER    MYPROG.CBL
15:43:00 USER    07 BLOCKS FREE
15:43:01 MONTR
15:43:01 MONTR   .
15:43:01 MONTR   .KJOB DSKB:COBJOB.LOG[10,1461]=/1/Z:4/B/VS:628/VL:200/VD:D
15:43:03 K-QUE   TOTAL OF 9 BLOCKS IN LPT REQUEST
15:43:09 KJOB    OTHER JOBS SAME PPN
15:43:12 LGOUT   JOB 24, USER [10,1461]  LOGGED OFF TTY103    1543  20-APR-72
15:43:13 LGOUT   SAVED ALL 43 FILES (855.  DISK BLOCKS)
15:43:13 LGOUT   ANOTHER JOB STILL LOGGED IN UNDER [10,1461]
15:43:13 LGOUT   RUNTIME 0 MIN, 07,14 SEC
15:44:07 LPMSG   LPTSPL VERSION 4A(141) RUNNING ON LPT2
15:44:15 LPMSG   JOB COBJOB FILE DSKB1:MYPROG.LST[10,1461] FOR [10,1461] STARTED
15:44:25 LPMSG   DSKB1:MYPROG.LST[10,1461] DONE
15:44:25 LPMSG   JOB COBJOB FILE DSKB1:MAP.LPT[10,1461] FOR [10,1461] STARTED
15:44:35 LPMSG   DSKB1:MAP.LPT[10,1461] DONE
```