INTRODUCTION TO DEC SYSTEM-10: TIME-SHARING and BATCH

THIRD EDITION

T. W. SZE

PROFESSOR OF ELECTRICAL ENGINEERING

UNIVERSITY OF PITTSBURGH

Printed in the United States of America

Library of Congress Cataloging in Publication Data:

Sze, T. W.
Introduction to DEC System -10

Pittsburgh, Pa. : Univ. of Pittsburgh

Library of Congress Catalog Card Number: 80-54311

CONTENTS

PREFACE OF THE THIRD EDITION

Completion of the Third Edition marked the tenth year since the book project first started. Materials of the First Edition were the results of organizing the class notes of a freshman course I developed and taught. The organization of the text was aimed in such a way that (1) materials were presented in several levels of depth so that a beginner can quickly acquire a basic skill, and (2) a subjective judgement was exercised in the relevancy of materials to the intended readers, who will use the computer as a tool in their fields but have no desire to become professional programmers.

The experience of using these materials, class notes and earlier editions of the book, seems to bear out this rationale. So the Second Edition simply updated the progress in the DEC-10 hardware and softwares. However, during the past few years, there have been very significant changes in the computer maturity of our student body in Engineering. High school instructions, microcomputer projects, hobby electronics all have contributed to this. As a result, the changes in the Third Edition involve a great deal more than just updating the changes and progress in DEC-10. Specifically:

(1) Three chapters in PIL and BASIC languages are deleted, and they are replaced by chapters in Program Debugging, Modeling and Simulation, and Computer Graphics. Only a summary of PIL is retained as an appendix in the Third Edition.

(2) The book is now sharply directed to the goal of using the computer as a system. Therefore, although FORTRAN is the fundamental programming language, the book is not intended to be a programming manual. At the School of Engineering, this book was used in a second course, after the students have their initial instructions in the FORTRAN language.

(3) In using the computer as a system, the book aims to remedy the most neglected and yet the most important phase of computer processing, namely, the debugging of a program. Many people still consider that as an art, and cannot be taught. The Third Edition makes a serious attempt on the study of program debugging. An entire chapter is devoted to that subject.

(4) The chapter sequence is re-arranged so that the front part of the text would be appropriate as a text, and the latter part as a reference. In addition, exercise problems have been added to help readers sharpen their skills.

As in the last two editions, I am most indebted to my family. In spite of their own busy professional and college schedules, my daughter Deborah and my son Daniel found time to read the manuscript and made both technical and grammatical suggestions. My wife Frances, beside being understanding and encouraging, took charge of style review and proof reading, and made suggestions that increased the readability immeasurably. Students and colleagues, too numerous to list, have been most helpful; their questions, suggestions and ideas were indispensable. Finally, I wish to acknowledge the Computer Center at the University of Pittsburgh for providing the facilities and environment that made this book possible.

November 23, 1980
Pittsburgh, Pennsylvania

T. W. Sze

CHAPTER 1

INTRODUCTION

1.1 Batch Processing versus Time-Sharing

Once upon a time, when a computer user wanted to run a program, he would have to go through the following steps:

(1) The user submitted his program and data deck to the Computer Center.

(2) The decks of cards submitted by different users were stacked together to form a batch, each deck with its proper identification. All jobs in one batch were then executed in one "run", hence the name "batch processing". The information on the punched cards in a batch were first copied into a reel of magnetic tape by means of a small and relatively inexpensive computer. The reason for this was that the card-input to the main computer was a slow and therefore expensive process.

(3) The magnetic tape so prepared became the input medium to the main computer. At the scheduled time, the jobs in the batch were run and the outputs (printouts, cards, tapes, etc.) were obtained. Sometimes the outputs were recorded on another reel of magnetic tape; then output printing may be done off-line so as not to slow down the computer operation.

(4) The outputs were returned to a designated place of the Computer Center for the users to pick up.

During the execution of a job in one batch, such as to compile and execute a FORTRAN program, each job had the undivided service of the entire computer, with all of its memory, input and output devices, supporting services and library routines. When the next job entered the computer, that job in turn received the total service of the computer for the duration of the job execution, however brief.

Economics and efficiency considerations have led to the techniques of multi-programming in batch processing, so that several programs may be executed interleavingly when devices required for execution are not in demand at the same time, or if a priority of queuing can be clearly established.

From the point of view of economy and machine efficiency, batch processing indeed represents the best computer utilization because it can serve a maximum of users within a given span of time. The prime consideration is then

the efficient usage of computing resources, even if it is done at the expense of efficient usage of user resources. Therefore, from the users' point of view, batch processing has many limitations.

The time interval between submitting a card deck to the Computer Center and retrieving the results, called the turn-around-time, may vary from several minutes to several days. Such long intervals are most frustrating to a user during the program preparation and debugging stages. A minor error of an incorrect punctuation mark in a program can cause a delay of hours or days. Once the grammatical errors are removed, it still requires many successive runs to remove logical errors. These consecutive runs cannot be hastened because the second run depends on the first, the third depends on the second, and so on. That made the debugging stage the most tedious and frustrating part of the program development.

Thus the early work in time-sharing research was motivated by correcting the tedious and frustrating process of debugging in the batch mode of operation. The reasoning that led to time-sharing was that the human responses and the output device responses are very slow in comparison to the logic and computing speeds of the computer; hence, it may be possible to switch the computer from one user to another and still seem to maintain a continuity at each user's station.

In the time-sharing mode of operation, a computer will service the jobs entered at remote terminals by sequentially giving a short period of time, called a time slice, to each job. Once that time slice is exhausted, that particular job is returned to the end of the queue to wait for another turn. In the meantime, a monitor program will perform the necessary bookkeeping and housekeeping tasks so that when that job receives a time slice again from the computer, the execution will pick up where it was left off.

From early 1960's when the time-sharing system concept was first developed, this mode of operation for a computer became widely accepted as an augmental mode of operation. However, before very long, it became quickly apparent that the major benefit is not the reduction of programmer frustration, but an entirely new dimension of problem-solving not possible before, utilizing a high degree of interaction between man and machine as a team. The language processors and programs may then be so designed that during the execution of a program, not only can error messages be sent to the user to aid his debugging, but also the user is able to modify his problem solving tactics and procedure as he sees the partial results along the way. It is possible then to design programs subject to modification by the user during execution time to adapt themselves to the condition of the problem.

Figure 1.1 shows a typical time-sharing computing system hardware organization. The configuration consists of a computer located at the Computer Center and the communication control, transmission and receiving equipment to connect the computer with the users at the remote terminals. The data line multiplexer and controller is used to control and direct the schedule of time-sharing activities. At the user's terminals, each terminal is connected to a data set or modem (modulator-demodulator) that converts the output signals from the terminal into a form suitable for transmission by the communication channel. The communication channels are usually commercial telephone networks, although in many cases telegraph lines and microwave channels are also used. The data set or modem at the receiving end re-converts the transmitted signals back to a form suitable for processing by the computer circuits.

It is also of interest to note that the remoteness of remote terminals is only limited by the quality and the economy of the communication equipment. At the present level of communication technology, it is commercially practical for

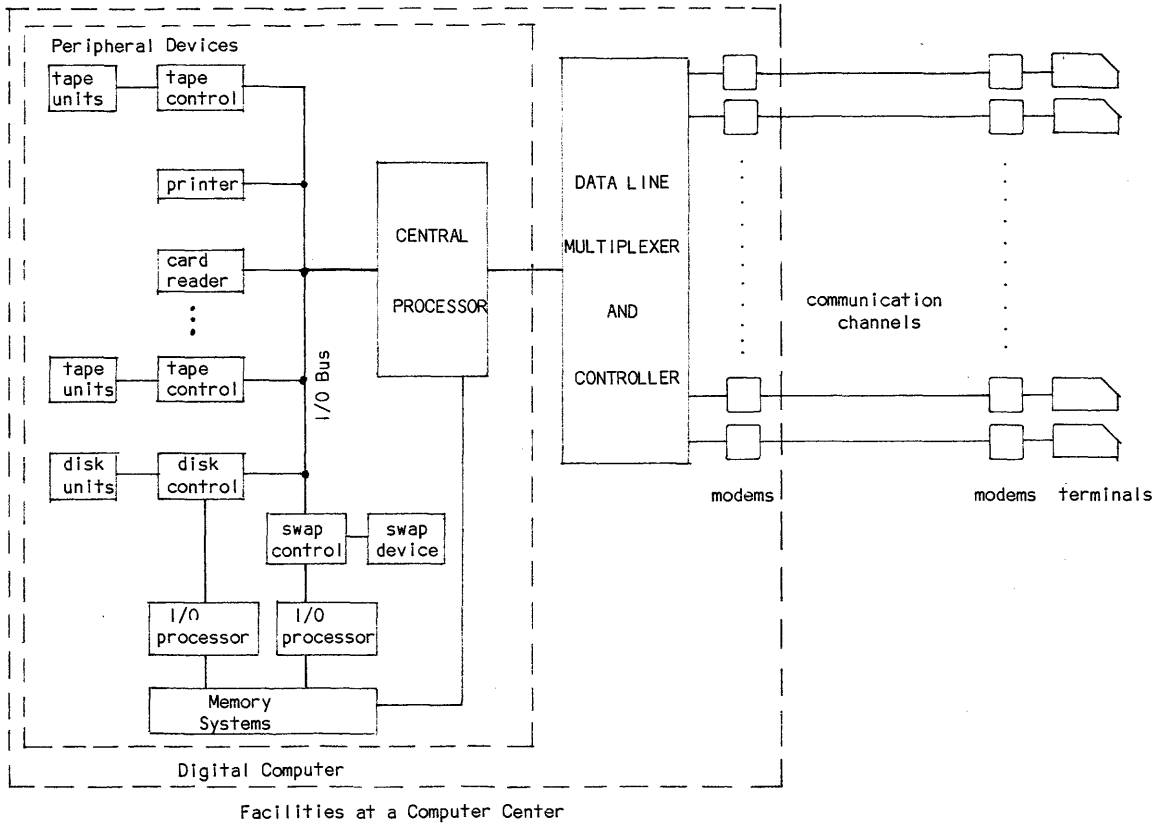


Figure 1.1 A Typical Time-Sharing System Hardware Configuration

a large, centralized computer to serve on demand users scattered over a wide area over the world. Thus through the time-sharing use of computer, an entirely new "utility" has emerged, just like electricity, gas, or water, to provide the users with the computer services when independent ownership of these services may be out of reach economically to these users.

1.2 Time-Sharing System at Pitt

University of Pittsburgh is one of early pioneers in the development of time-sharing computer system. Through a federal grant in 1965, the time-sharing facilities for the University community were established, using an IBM 360/50 system. Much of the software supporting facilities was developed in the subsequent years, resulting in a system then known collectively as the Pitt Time-Sharing System, or the PTSS.

In 1971, the time-sharing computer at the Pitt Computer Center was changed to a multiple PDP-10 system of Digital Equipment Corporation. This system has been upgraded and expanded several times, and the present configuration is a dual DEC-1099 system. Figure 1.2 shows the configuration of the system.

As in many similar environments, the current software system is a combination of vendor-supplied software and self-developed facilities. The readers are referred to the list of references at the end of this chapter for details of language processors and other software subsystems.

The software system of the time-sharing system contains, in addition to the language processors, a group of service routines. The most important one for the time-sharing operation is the executive system, also called a supervisor or a monitor. It is a master program which exercises an overall control on the time-sharing activities. It performs the scheduling of users from the queue, provides users with proper language processor and peripheral facilities as requested by the users, keeps an account of charges, and provides a variety of service functions.

Because of the control it exercises, the monitor is the highest-ranking program in the software system. The monitor controlls and dispatches a group of processors, collectively called the CUSP (Commonly Used System Programs), among which are the language processors such as BASIC and FORTRAN. In turn, under the control of each CUSP is a subgroup of routines for the execution and/or interpreting of the instruction set of the CUSP. Thus the software system has a distinct hierarchy structure, and this is shown in Figure 1.3.

There are several points regarding the software system structure worthy of note:

(1) There are three levels of hierarchy: the monitor level, the CUSP level, and the sub-CUSP level. The monitor level is the highest.

(2) It is a common practice in a time-sharing system for the computer to supply a prompting symbol through the user's terminal to indicate that the computer is ready to accept a command or input data. In the time-sharing system of DEC-10 system, different hierarchies use different types of prompt symbols:

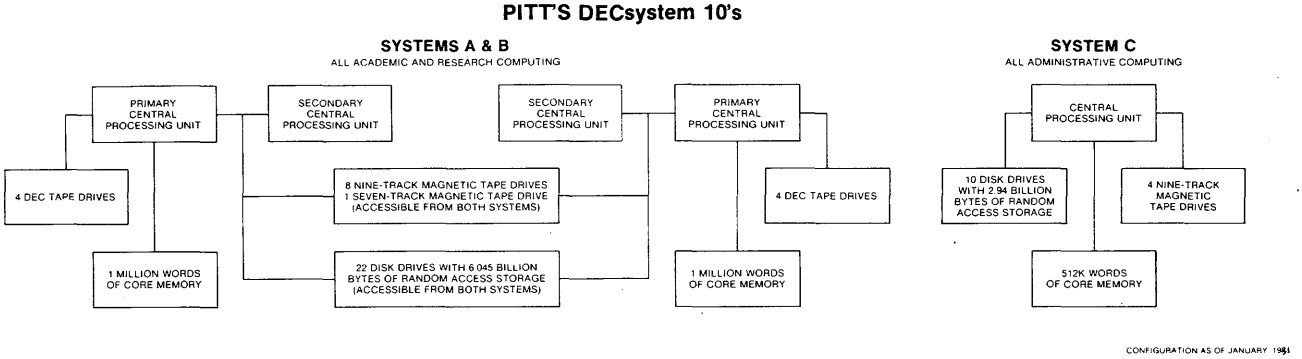


Figure 1.2 Configuration of a DEC-1099 System at Pitt

(Each central processing unit is a PDP-10 CPU)

Reprinted by permission, Reference 6
Computer Center, University of Pittsburgh, Pittsburgh, Pa.

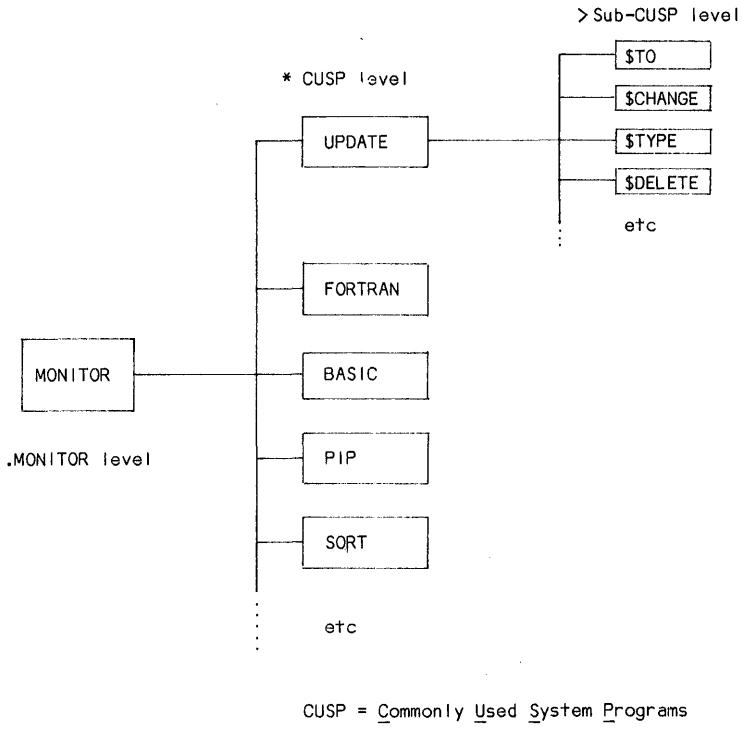
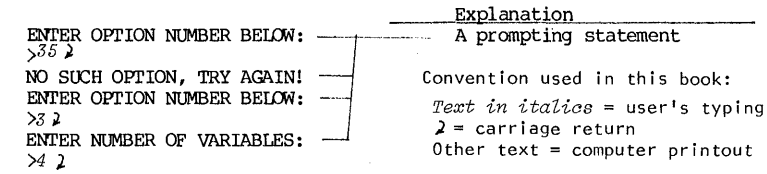


Figure 1.3 A Typical Time-Sharing Software System Organization

Prompt Symbols	Explanations	Hierarchy Level
.	A period	Monitor level
*	An asterisk	CUSP level
>>	Double ">" signs	
>	Greater-than sign	Sub-CUSP level
?	Question mark	

(3) When such a prompt symbol appears on the user's terminal, the computer is ready for a command or information, and the user must type in a command or data and terminate the typing with a carriage return. However, when

a program contains a number of such command/input breakpoints, it becomes difficult for the user to keep track exactly what command/input is expected at each breakpoint. It is therefore necessary for the program to be designed so that a statement of instruction or prompting message is printed on the terminal at each breakpoint in order to guide the user. The following shows a typical example: (user's typing in *italics*)



Thus, the combination of the prompting statement, the prompting symbol and the user's response constitutes a man-machine interaction, and is referred to as a man-machine dialogue. Programs using extensive dialogues to guide the users are called conversational programs.

(4) It is not possible to transfer directly from one CUSP-level language processor to another without first returning to the monitor. This transfer can be made conveniently by providing a special control key on the remote terminal keyboard. See Section 1.6 for the function of various keys on the keyboard.

1.3 Computer Service

The computing facilities in an academic institution are generally provided to serve a combination of instruction, research and administration functions. When the facilties are shared by different users for different functions, it is necessary to establish rules and regulations so that the resources may be most efficiently and equitably utilized. While it is outside the scope of this book to enumerate these rules and regulations, it is important for every user to be familiar with them. These include such matters as application procedures, allocation of computing time and resources, restrictions placed on the computing services, fiscal arrangements, ethical and legal stipulations regarding security, propriety and relevance of work using the computing resources, and policy on computer abuses.

Application procedures are generally defined by the Computer Center to determine the eligibility and extent of computer usage of an applicant. The application requires certain pertinent facts and the usual authorizing signatures. Readers are referred to their respective Computer Centers for current procedural details.

When an application is accepted, the applicant is assigned a pair of identifying numbers:

[m , n]

where m = a 6-digit (octal) project number, and
n = a 6-digit (octal) programmer (user) number.

The combination of these two numbers, referred to as the project-programmer numbers, is often abbreviated either as PPN or as P,PN. Note that a PPN is usually enclosed in a pair of square brackets.

REMOTE TERMINALS

1.4 Communication with the Computer

A remote terminal is used as an input or output device at the control of the user. Generally, it is a typewriter-like device with a keyboard, a typing or displaying element, and the interface between the user and the system. The performance of the remote terminals depends to a large extent upon the communication linkage between the terminals and the computer. Hence, some of the basic concepts and terminology will be described here to aid the understanding of a time-sharing terminal.

(1) Transmission line

Depending on the modes of information transmission, the transmission lines, also called channels, are classified as simplex, half-duplex, and full-duplex. A simplex channel can transmit information in one direction only. A half-duplex channel can transmit information in either direction, but only in one direction at a time. A full-duplex channel can transmit information in both directions at the same time.

Depending on the physical connections, transmission lines may be classified as dedicated, shared, hard-wired or dial-up lines. A dedicated line or channel is one assigned for the exclusive use of the terminal. A shared line is one assigned to the use of several terminals. A hard-wired line connects physically from the terminal to the system. A dial-up line is a shared line using the commercial dial telephone network for connection.

(2) Information code

Information to be transferred externally between a terminal and a computer on the transmission line is represented by character sets consisting of alphabetic characters, both upper and lower cases, numeric characters, punctuation marks and special characters. In addition, signals representing control action of transmission and processing are coded into "control characters". These information characters and control characters may be coded into a series of binary digits (called bits) so that information may be transmitted and processed by the computer and the terminals. Several systems of codes are in use. The code format used in most U.S.-made non-IBM machines, including the systems at Pitt, is the ASCII* code, which encodes 128 characters into 7 binary digits. Table 1.1 shows the ASCII code assignment of characters, where the code assignments are given in octal numbers. For example, the upper case letter "A" is coded as octal 101, or actually as 7-bit binary representation of 1000001.

Note that the character set shown in Table 1.1 is the ASCII information-character set, which is a subset of the complete ASCII code of 128 characters. The 32 characters not shown in Table 1.1 are all control characters. With ASCII code, the words PITT and Pitt are then transmitted respectively as:

```
1010000 1001001 1010100 1010100    (PITT)
1010000 1101001 1110100 1110100    (Pitt)
```

---

*Acronym for American Standard Code for Information Interchange, usually pronounced as "AS-KEY".

Character	ASCII 7-Bit	Character	ASCII 7-Bit	Character	ASCII 7-Bit
Space	040	@	100	'	140
!	041	A	101	a	141
"	042	B	102	b	142
#	043	C	103	c	143
$	044	D	104	d	144
%	045	E	105	e	145
&	046	F	106	f	146
'	047	G	107	g	147
(	050	H	110	h	150
)	051	I	111	i	151
*	052	J	112	j	152
+	053	K	113	k	153
,	054	L	114	l	154
-	055	M	115	m	155
.	056	N	116	n	156
/	057	O	117	o	157
0	060	P	120	p	160
1	061	Q	121	q	161
2	062	R	122	r	162
3	063	S	123	s	163
4	064	T	124	t	164
5	065	U	125	u	165
6	066	V	126	v	166
7	067	W	127	w	167
8	070	X	130	x	170
9	071	Y	131	y	171
:	072	Z	132	z	172
;	073	[	133	{	173
§	074	\	134	|	174
=	075	]	135	}	175
↑	076	↑	136	~	176
?	077	←	137	Delete	177

The code assignments of octal numbers from 000 to 037 are for control characters, and are normally of no concern to an average user. However, certain control characters pertain to printer control, and it will be useful to know their code assignments. These are:

Line Feed	012	Form Feed	014
Vertical Tab	013	Carriage Return	015
	Horizontal Tab 021		

Table 1.1 ASCII Character Set

All numbers in octal codes.

In an actual transmission, each ASCII-coded character is packed together with additional bits that perform functions of synchronization (START and STOP of each character), error-checking (parity bit), and filler or dummy bit (to allow slower mechanical components to catch up with electrical and electronic components). The result is either an 11-bit group (for low-speed transmission) or a 10-bit group (for higher speed transmission) for each character transmitted.

Not all computers made in U.S. use the ASCII code. The IBM computers, such as System/360 and System/370 machines use a code system called EBCDIC (Extended Binary Coded Decimal Interchange Code) to adapt to its byte-structure (1 byte=8 bits). Hence, output media, such as magnetic tapes, are not compatible between ASCII-code machines and EBCDIC-coded machines without first going through a code conversion process. Because of wide-spread use of both code systems, such a code conversion routine is a part of standard service routines available at the Computer Center. For the same reason, a remote terminal wired to accept the EBCDIC code cannot be used in the DEC-10 system unless it is re-wired or it has a switchable option of code selection.

While the ASCII code has been adopted as the American standard for peripheral communication, it has shortcomings in certain particular applications. For example, the internal representation of a FORTRAN variable would be very awkward in a machine such as the DEC-10 with a 36-bit memory word format. Since the standard FORTRAN defines a variable name to contain one to six characters, an ASCII-coded six-character FORTRAN variable name will require 42 bits or 2 memory words for its storage, a rather inefficient usage. As seen in the Table 1.1, if we forego the difference between the upper and the lower cases of alphabetic characters, we can omit the right-hand column in that table. This would reduce the character set to only 64 characters. Since each of the 64-character set may be uniquely defined by a coding scheme of six binary digits, this results in a Sixbit Code. With each character code only six bits long, a six-character FORTRAN variable name can now fit snugly into a single 36-bit word. In this coding system, any lower case alphabetic character, when encountered, will be automatically coded as its upper case equivalent. The code assignment of each character in the Sixbit Code will not be tabulated here, but will be given later in Chapter 3 (FORTRAN-10) where its reference will be more relevant. The derivation of the Sixbit Code of a character from the 7-bit ASCII code may be obtained simply by dropping the second bit (counting from the left). For example, the letter "A" is coded as 1000001 in ASCII code, and is 100001 in Sixbit. Alternately, the Sixbit Code can be "computed" from the ASCII code by either of the following algorithm:

```
(SIXBIT)  = (ASCII) - 040      in octal arithmetic
(SIXBIT)  = (ASCII) + 040      in octal arithmetic
```

and then retain the two least significant octal digits.

Thus, the letter "A" is coded as octal 101 in ASCII, and as octal 41 in Sixbit.

(3) Speed of transmission

The speed of transmission of the signal is measured by the rate of transmission in signals per second, expressed in bauds*. In binary transmission, each signal contains one bit of information, and consequently the speed of signal transmission is numerically the same as the speed of information transmission. Thus a 300-baud line will transmit information at a rate of 300 bits/second. However, in polyphase modulation, each of the four predetermined phase-shifts represents two bits of information, and a 300-baud line will transmit information at a rate of 600 bits/second. Capability of commercial transmission services, such as telephone or telegraph lines, ranges from 100 to several hundred thousand bauds. The maximum capability of a "voice grade" dial-up telephone line is about 2000 bauds.

In a time-sharing system, information transfer may be initiated or terminated at the terminal. The ASCII coded 7-bit signals arriving at or departing from a terminal are packed with additional bits to perform functions of synchronization and parity error checking. The result is an 11-bit group for each ASCII character for 110-baud transmission, or a 10-bit group for 150 or 300 baud rate transmission. These transmission speeds are used to match the terminal output speed of 10, 15 or 30 characters/second respectively.

Since the remote terminals generate and receive information at relatively low speed, the capability of the transmission line is hardly taxed. Consequently, various line-sharing techniques are available, one of which involves the use of a concentrator. A concentrator is usually a minicomputer which collects information from several terminals in the area at a low speed, and then packs them and re-transmits. In the reversed direction, a concentrator receives information and distributes them to different terminals.

1.5 Description of a Remote Terminal, the DEC LA36 DECwriter

For several decades, the most commonly used communication terminals have been the Automatic Send-Receive Teletypewriter Set (ASR), model 33, 35 or 38. These are called ASR33, ASR35, and ASR38, and in most cases, simply Teletype ®. In fact, the standard abbreviation for terminal-like device in a computing system has been uniformly taken as TTY.

Rapid recent advances in technology have produced new generations of remote terminals. Relay circuits were replaced by transistorized circuits, which in turn are being replaced by microprocessors or microprogrammed controllers with semiconductor memory. Mechanical components are improved so that they are lighter and move faster. Clumsy typing heads with embossed characters are replaced with matrix wire impact printing, thermal or electrostatic non-impact printing. While the technological advances have made new generations of terminals lighter, faster, less expensive and more reliable, the basic operating principles procedures remain essentially unchanged, thanks to the tremendous steadying effect of Teletypes as the industry workhorse over the last three to four decades. It is therefore possible in the present discussion of remote terminals to deal specifically with one particular terminal and still retain generality of our discussion. It also means that although this presentation pertains to only one model of terminal, extension of the discussion

---

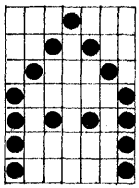
*Named after the French inventor of the telegraph code, Jean-Maurice-Emile Baudot, 1845-1903

® Registered trade mark, TELETYPE Corporation, Skokie, Illinois.

to another make or model would be no problem. This is why we will now concentrate on one particular terminal, the DEC LA36 DECwriter, for the subsequent discussion.

A remote terminal contains generally four major parts. They are the keyboard unit, the printer unit, the print control unit, and the call control unit. A simplified block diagram, with the important components within each part, is shown in Figure 1.4. The arrows in the diagram show the direction of signal flow and/or control when the terminal is connected to a computer. Its operation can be described briefly in this manner:

When the user strikes a key on the keyboard, say the upper case "A", the keyboard electronics encodes it into the ASCII code of signal 1000001. These electric signals are sent to the transmitter unit, in which additional start-bit, stop-bit, parity-bit and filler-bit (if needed) are added. The communication electronics in the transmitter transform these signals into modulated audio tones, which are transmitted serially through the interface to the computer over a transmission line. At the computer end, a buffer (or temporary) memory accepts the character after checking over any transmission error, repacks the character with start-, stop-, parity- and filler-bits, and re-transmits back to the terminal. When it reaches the terminal, the receiver demodulates the signals by removing the audio carrier, checks for any transmission error, and deposits the 7-bit ASCII code of "A" in the buffer memory.



When the printer unit is ready to accept a character, controlled by the printer control unit, the ASCII code inputs are sent to the character generator ROM (Read Only Memory, a semiconductor memory chip) which produces seven one-or-zero signals simultaneously 7 consecutive times. Each 7-signal group, after amplification, selectively actuates by solenoids vertically arranged wires to strike an inked ribbon, leaving a vertical column of selectively placed dots in that column.

This is repeated seven times, each time with the print head moving slightly to the right, and each time producing a different vertical pattern. The result is shown here. This is called a 7x7 dot matrix print.

It is interesting to note that the signals generated at the keyboard take a circuitous route before finally printed on the terminal printer. In fact, what is printed is actually what the computer thought the user has typed. This is a clever way of involving the user as a part of error-checking system, and is a standard feature in time-sharing system called echo print.

The individual parts of the DECwriter will now be described next:

(1) Print unit

The print unit is the receiving component of the terminal. It consists of seven vertically arranged print wires actuated by seven solenoids, which in turn are controlled by the character generator ROM as explained before. Other useful information about the print unit are as follows:

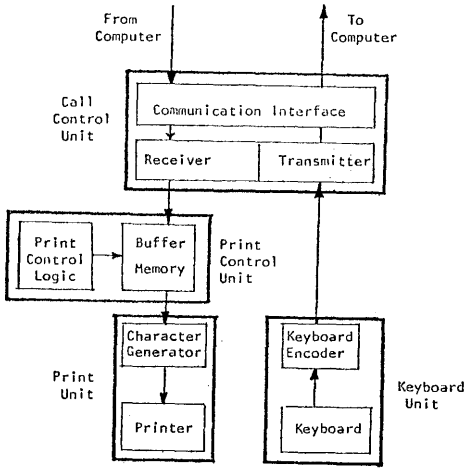


Figure 1.4 Block Diagram of a Typical Remote Terminal

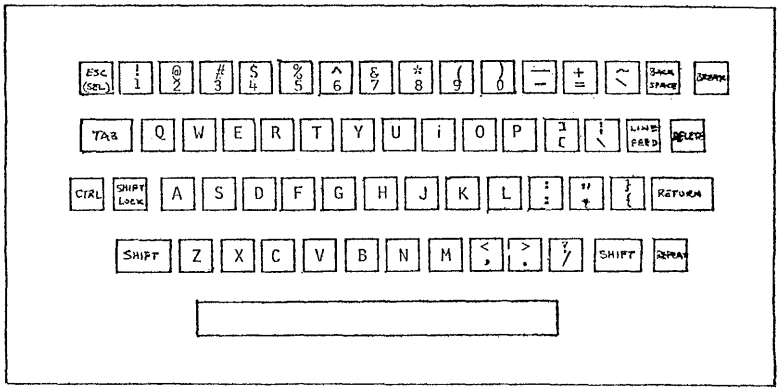


Figure 1.5 Standard ANSI Keyboard Layout

Paper size:	3" minimum, 14" maximum width
Print field:	132 characters maximum
Print spacing:	10 character/inch horizontal, 6 lines/inch vertical spacings
Print characters:	96 upper/lower case ASCII 7x7 dot matrix (0.07x0.10 inch)
Print speeds:	switch selectable at 10, 15 or 30 characters/second with 60 char/sec catch-up mode*

(2) Print control unit

The print control unit contains a buffer memory that accepts ASCII character codes received and a control logic unit which is a microprogrammed controller. Under the control of the microprogram, characters in the buffer are presented to the character generator on a first-in/first-out basis. The microprogram activates the carriage servo system and the print head system to control the mechanical movements. It also detects signals and actuates mechanisms such as line feed to advance the paper, ringing the bell for an error, etc.

(3) Call control unit

The call control unit consists of an asynchronous receiver-transmitter and a communication interface. It initiates, accepts, controls and completes the incoming and outgoing transmission of information.

(4) Keyboard

The keyboard is the information-sending component of the terminal. The mechanical linkages and electrical contacts translate the key action into a group of electrical signals. The arrangement of keys on the keyboard resembles that of a conventional typewriter with additional special features. These are discussed in a later section in this chapter.

Those terminals called ASR's (Automatic Send-Receive Sets) contain, in addition to the four units mentioned above, one of the following auxiliary input/output units: paper tape reader/punch, or tape cassette player/recorder, or floppy disk with read/write electronics. These serve as storage media for the terminal.

There are several switches placed adjacent to the keyboard which allow a user to power-up, select the transmission rate, and choose on-line or local operation. In local operation, a terminal will function as a typewriter, allowing a user to add information to the printout. It also permits maintenance work and testing of a terminal without disturbing the computer. For ASR-type terminal with either paper tape, digital cassette or floppy disk, the LOCAL position permits the ASR to be used as an off-line input/output device for such task as preparing, editing, reproducing and printing paper tapes, cassette tapes, or floppy disks.

---

*While the time-consuming action of carriage return, tab or line feed is taking place, characters received are stored in the buffer. When mechanical action is finished, characters in the buffer will empty into the printer unit at 60 char/sec catch-up speed.

1.6 The Keyboard

The keyboard arrangement of the DECwriter follows the ANSI (American National Standard Institute) standard. It has a format very similar to the conventional typewriter. Figure 1.5 shows a keyboard of the DECwriter.

(1) Alphabetic characters

Key positions of alphabetic characters are identical to those on a conventional typewriter. Both upper and lower cases are available. However, transmission of alphabetic characters are generally done in upper cases, unless specifically commanded to transmit as lower cases. Thus, pressing an alphabetic key without the shift key will transmit and print an upper case letter.

(2) Numeric and special characters

The character set on the DECwriter keyboard consists of the following:

Alphabetic:	A B C D E F G H I J K L M N O P Q R S T U V W X Y Z
Numeric:	0 1 2 3 4 5 6 7 8 9
Special:	+ - * / ( ) = " $ % ' @ \ . , ; : ? [ ] < > { } _ |

(The underscore symbol _ is replaced with the left arrow symbol ← on certain keyboards.)

(3) Control keys

Certain special keys perform control functions:

a. LINE FEED — This key will cause the terminal paper to advance one line. When the terminal is operated on LOCAL, the carriage return does not automatically advance the paper, and the LINE FEED key must be pressed to do it.

b. RETURN — This key will return the print head and carriage. When the terminal is on line, returning the carriage return signifies the end of a unit of information, for example, an instruction to the computer. The computer will automatically respond with a line-feed control signal to advance the paper.

c. DELETE — This key permits the correction of typing errors on a line if the carriage has not yet been returned. This key is marked as RUBOUT on some older keyboards. When the DELETE key is pressed successively for n number of times, the last n characters typed (including spaces) will be deleted. As a signal to the user which characters are being deleted, the terminal will print out the deleted character each time the DELETE key is pressed. Also, before the first deleted character and after the last deleted character, a back slash "\" is printed. Thus the pair of back slashes serves as delimiters bracketing the string of deleted characters.

For example, if the following has been typed on the terminal:

```
FOUR SCORE AND SEVIN YE
                      ↑
                      |
                      Carriage position when mis-spelling
                       in SEVIN is discoverd.
```

In order to delete the five character "IN YE", five successive DELETEs are required. Notice that a space or blank is also considered a character. To correct the typing, the user will DELETE five times and then retype the corrections. On the printout at the terminal, it will appear like this:

```
FOUR SCORE AND SEVIN YE\EY NI\EN YEARS AGO, OUR FATHERS ...
                       ||||  |
First DELETE ----------+|||  |
 Second DELETE ---------+||  |
 Third DELETE ----------+|   |
 Fourth DELETE ----------+   |
 Fifth DELETE ---------------+
                             printout when resume typing
```

As shown in the example, a pair of back slashes brackets the deleted characters printed in the order of deletion (from right to left).

d. REPEAT — This key, when operated together with another character key, will cause a repetition of that character to be printed (for LOCAL operation), or a repetion of that character to be transmitted and echo-printed (for on-line operation). For example, when the REPEAT and K keys are pressed down together, a string of K's will be sent and printed as long as both keys are held down.

e. SHIFT, SHIFT LOCK — These keys have identical functions as those on a conventional typewriter, and will cause the upper case character marked on the key to be printed or transmitted.

f. ESC — This key, appearing on older keyboards as an ALMODE key, directs the computer to treat the next received character as a command. The precise meaning of the ESC-character combination is defined by the software system employing this function.

g. BACK SPACE — Depending on the software processor used at the time, the back space key either makes the last character sent to the computer available for deletion or correction, or makes it possible to overprint with a different character such as underscoring a certain text string.

h. TAB — This key will direct the computer to advance the print head to the next tab stop.

i. BREAK — Used for half-duplex transmission mode to interrupt reception of data from the computer. Ignored in ordinary full-duplex mode.

(4) Control characters

The key CTRL, when used together with an alphabetic character key, generates a code combination for control purposes. Such a combination of CTRL and alphabetic keys does not have any printing function, and therefore the computer will return an echo signal printed out on the user's terminal to inform him of the nature of the control function. The echo print has a format of "^" (a circumflex) or "↑" (an up arrow) followed by the character used, such as ^C or ↑C. These control characters will appear in this book frequently, and they will be referred to in several ways. For example, the control character C will be referred to as:

CONTROL-C
CTRL-C
or, ^C (or, ↑C)

Although there are 26 control characters, a beginning user need only be familiar with a few of them, and they are ^C, ^O, ^U, ^I, ^L, and ^R. Several other control characters, such as ^S and ^Q, will be explained at appropriate places where they are used.

a. CTRL-C (^C) The ^C key interrupts the program and returns the control to the system monitor. If a program execution is in progress, apply ^C twice or more to interrupt it. The first ^C stops the execution of the program, and the second one (and the succeeding ones) returns control to the system monitor. When the system monitor obtains the control, a prompt symbol (a period) is printed on the terminal, and the system awaits for a monitor command.

b. CTRL-O (^O) The ^O key suppresses the terminal output without interrupting the execution of a program. For example, when debugging a program, if you only want to see whether a program execution reaches the end, you can suppress all or specific parts of the put in order to avoid time-consuming printing of the results. Thus any time when output begins to appear, applying CTRL-O will suppress the remaining portion of that output.

c. CTRL-U (^U) The ^U key, applied at the end of one line of typing, will instruct the computer to ignore the entire line and therefore to perform the function of deleting that line. The system will respond with a carriage return and a linefeed, but no prompt symbol.

d. CTRL-I (^I) Terminal will tab to a pre-set column.

e. CTRL-R (^R) Terminal will re-type the current line.

f. CTRL-L (^L) This control character tells the computer to advance the paper to a new page. On a DECwriter terminal, it will advance the paper only 8 lines.

The keys for these functions are summarized in Table 1.2.

Special Key	Echo Print If any	Function
LINE FEED		Move paper up one line.
RETURN		Return the carriage.
DELETE (or RUBOUT)	/X	Delete character immediately before.
REPEAT		Repeat a character or a function.
CTRL-C	^C	Return to monitor mode.
CTRL-O	^O	Suppress current terminal output.
CTRL-U	^U	Ignore the current line input.
CTRL-I		Tab to a preset column.
CTRL-R		Retype the current line.
CTRL-L		Advance paper on terminal 8 lines.

Table 1.2 Function of Selected Special Keys

1.7 Other Types of Remote Terminals

The DECwriter terminal as described in the previous section is a keyboard printer terminal. The majority of remote terminals used in a time-sharing system are of this type. A variation of this type is the portable terminal, which incorporates in a single carrying case an acoustic coupler (for connecting the terminal to the computer by a telephone set), a keyboard, a printer and associated electronics. One ingenious product includes an acoustic coupler, a keyboard, and associated electronics, but no printer. It makes use of a conventional television set, and when combined, it becomes a time-sharing terminal.

As a result of rapid advances in MOS/LSI (metal oxide semiconductor and large-scale integrated circuits) technology, the size, weight and cost of electronic components and systems have been greatly reduced. These advances have caused rapid developement of other types of terminals, and they are briefly discussued next:

(1) Cathode ray tube (CRT) terminal

The convenience of a keyboard operating terminal is greatly enhanced if we use a cathode ray tube (CRT) terminal for the purpose of communication with the computer, preparation of programs and debugging. This is particularly useful if the user has an alternate means of producing hard copy as records.

A typical CRT terminal, also called a scope terminal, displays a subset of ASCII characters (such as upper cases of alphabet plus symbols). The display unit is similar to that used in an oscilloscope or television set. Other than the display unit and its control memory, it has the same organization as a keyboard printer terminal. Figure 1.6 shows a block diagram of a typical CRT terminal. If we compare this with Figure 1.4, we can see the obvious resemblance.

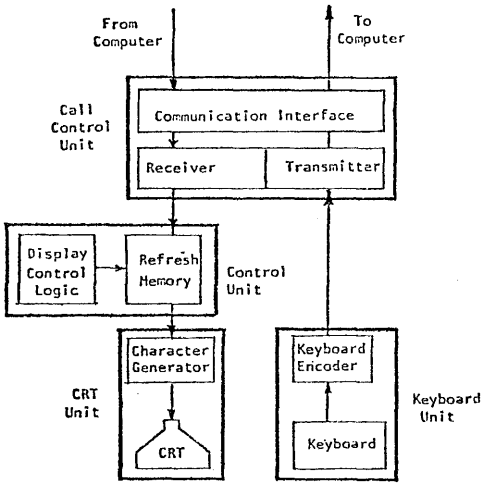


Figure 1.6 Block diagram of A CRT Terminal

The CRT produces an image by directing an electron beam against a phosphor-coated screen which emits light when struck by electrons. The control on the beam intensity can turn the beam completely off, thus allowing no electrons to strike the screen, a process called blanking. Positioning the beam in a CRT terminal is usually done by the raster scan method. The beam is first positioned at the upper left corner of the screen; it then moves across the tube face, producing a straight line. The beam is blanked while returning to the left at a level one line lower. The blanking is turned off, and a second line is traced. This is the same method used in a commercial television set that scans 525 lines/frame and at a rate of 1/30 second per frame. Forming characters on screen is very similar to the dot matrix print of a keyboard printer terminal. The scan scheme will position the rectangular area within which the character is displayed. The character, through a character generator ROM (read-only-memory), formulates a 5x7 dot matrix, with dots emitting light when the electron beam strikes the tube phosphor. Typically, the light-emitting period is very brief, ranging from microsecond to millisecond range. Therefore, a CRT using the scan method requires a refresh memory that stores the display and re-display at a refresh rate large enough to provide a constant intensity image and to eliminate flicker in the image.

Most CRT terminals are also teletype-compatible, and they are often interchangeable with keyboard printer type terminals. With no carriage, a CRT terminal is provided with a cursor, which may blink on and off to indicate the current position of the beam. The associated cursor control enables the user to move the cursor up, down, left or right, or to erase the screen. Unlike the teletype, data rolling off the top of a CRT screen are lost to the user. The operations of a CRT terminal and a keyboard printer terminal are very similar. A person familiar with the operation of a keyboard printer terminal should have no problem with CRT terminal operation.

(2) Graphics terminal

This is a terminal which maintains the capability of displaying not only characters, but also arbitrary figures. All of the man-machine interaction

previously described are retained, and the interaction is expanded to include the clarity of graphics.

When a graphics terminal displays characters, it emulates a CRT terminal, and outwardly it operates just like a CRT terminal. When a graphics terminal operates in the graphics mode, it provides both control of beam position and blanking. In the position control, the beam is deflected from a current position to another. If the blanking is on, only two end points are shown on the screen. If the blanking is off and if the terminal is equipped with a "linear interpolation vector generator," the electron beam will trace a straight line. Repeated programmed positionings of the beam, with blanking on or off as required, will produce a line drawing.

Graphics terminals normally utilize a cathode ray tube display, but some low cost units use a storage tube to retain the data which does not require a refresh memory. The disadvantage of the storage graphics display is that dynamic display and removing graphic information are not possible: any subtractive change of displayed data requires first an erasure of the entire image, and then a reconstruction of a new image, an event that will take at least half a second. Thus a storage tube may display at a maximum rate of about 2 frames per second, not a satisfactory speed to depict motion. On the other hand, graphics terminal using refresh memory imposes a heavy burden of memory and software support for its image generation and constant refresh. The heavy requirements of memory and software usually call for a minicomputer to provide the support.

(3) Intelligent terminal

For years, manufacturers have been offering terminal systems with fixed functional capability. For example, a terminal designed to be compatible with the IBM systems, which use a different character coding system (EBCDIC Code), is not compatible to a system using the ASCII code unless extensive re-wiring is done.

The rapid recent advances in MOS/LSI technology have now made it possible to incorporate microprocessors and memories, which greatly expand the flexibility and capability of a terminal. Instead of a simple function of transmitting and receiving data or programs, a terminal may now have additional processing power. Acquiring such additional processing power within the terminal is referred to as "making the terminal more intelligent", and therefore the name "intelligent terminal." Quite predictably, a terminal without additional built-in intelligence is called a "dumb terminal."

Intelligence in a terminal may take on many forms. It ranges from the simple ability of changing operating characteristics of the terminal to the power of a full-scale microcomputer. Intelligent terminals therefore are able to emulate many different communication line procedures and codes, so that a terminal may be coded to adjust to an existing line protocol and procedure. For many other various functions, the terminal may be tailored to suit the need of the particular user or industry segment by providing specific software for the intelligent terminal. For example, an editing program may be installed in the intelligent terminal so that the terminal becomes a word-processing machine. Word-processing tasks may then be carried out without loading down the central computer. Another example is an intelligent graphics terminal where the graphics are processed by a built-in graphic processor in the terminal. Again, in this way, the central computer will not be loaded down with detailed chores.

The main disadvantage of an intelligent terminal, at the time when the third edition of this book is being prepared, is its cost, although the gap is rapidly narrowing. In applications where only simple functions are required,

dumb terminals are more cost effective. In time, the difference in cost will become insignificant, and the intelligence of the intelligent terminal will be greatly expanded. The experiences of the hand calculator industry can very well be repeated in the remote terminal industry within the next decade. At the present time, the applications have been limited to such areas as point-of-sale credit authorization, bank teller systems, stock brokerages, airline reservation systems, hospital admissions, etc., where distributed data processing is highly desirable.

1.8 Sign-On at the Remote Terminal

Once a user has a valid pair of ID numbers (the PPN) and has a valid password, he may now sign on at any remote terminal by following the procedure outlined below:

Hard-Wired Units	Dial-Up Units
(1) Turn on switches. Press C if there is no prompt symbol ".". After the prompt "." appears, type "I" (for INITIATE) and the following lines will be typed out on the terminal:	(1) Turn on switches and dial the computer number.* If the line is busy, there is a usual busy signal. When the call gets through, a high-pitch tone can be heard. Place the phone set on the seat of the acoustic coupler. Wait until the READY or CARRIER light comes on, type C, and the following two lines will be typed out on the terminal:

```
PITT DEC-1099/A 63A.41B 15:36:41 TTY43 system 1237/1240
PLEASE LOGIN OR ATTACH
```

where "1099/A" indicates System A, "63A.41B" the monitor version, "15:36:41" the time of the day in 24-hour clock, "TTY43" the line number assigned. If "1099/B" appears instead of "1099/A", it means the user is in touch with System B. If the user finds himself in a wrong system, he requests a change by typing:

TTY SYSTEM B	or	TTY SYSTEM A

after the prompt symbol.

(2) Type the monitor command after the prompt symbol:

LOGIN m,n ↵

or

LOGIN m/n ↵

where m = project number, n = programmer number,
↵ = carriage return.

The difference between "m,n" and "m/n" in the two monitor commands is that the latter form will suppress the message of the day from the Computer Center when the sign-on procedure is completed. It is possible that you have seen the message several times already, and may not care to read it another time.

---

*For University of Pittsburgh users, dial (412) 621-5954.

The carriage return is a standard control signal to indicate to the computer the termination of a line, a command or a message. To avoid cluttering the text and to relieve the typing problem, the carriage return symbol " " will be used only in Chapter 1. For the remainder of the book, the readers should assume that there is always a carriage return at the end of every line.

(3) Enter the password when requested. The password will be entered in a non-print mode, and the typed password will not appear on the terminal. This is to maintain the security of the password.

If the entered password is an incorrect or invalid one, the system will respond with an error message and a request for the PPN. After supplying the PPN again, another password request will be made by the computer. The user has five chances to sign on correctly. After that number of unsuccessful trials, the job is killed, and the user must restart the entire procedure to sign on.

If the password is found to be valid, the system will respond with information on the status of the project, the last sign-on time and date, the time of day, and the "message of the day" from the Computer Center. The last item may be suppressed if the user uses the LOGIN command with the m/n specification.

After all preliminary reports are finished, a prompt symbol "." is printed on a new line, and the computer pauses and waits for input. The user is now connected to the computer at the monitor level, and the sign-on procedure is completed.

The following two cases are examples of sign-on. Explanatory remarks are also given along with the remote terminal printout. As used throughout this book, those lines entered by the users will be in *italics*:

Printout on Terminal	Remarks
.*INITIATE*)	INITIATE command
PITT DEC-1099/A 63A.41B 16:19:17 TTY43 system 1237/1240	Computer's response
.*TTY SYSTEM B*)	Request System B
PITT DEC-1099/B 63A.41B 16:19:50 TTY43 system 1237/1240	
.*LOGIN 115103,320571*)	Sign-On command
JOB 35 PITT DEC-1099/B 63A.431B TTY43 Wed 7-May-80 1619	
Password: (*Your password*))	Supply password
Last login: 7-May-80 1617	
Usage ratio: 22.13 Units used: 33.5	Password valid
SYS B DOWN 0000-0800 MON MAY 12 FOR REGULAR HARDWARE MAINTENANCE	
SYS B DOWN 0000-0300 TUE MAY 13 FOR REGULAR SOFTWARE MAINTENANCE	
DUE TO HARDWARE PROBLEMS THE ARRAY PROCESSOR WILL BE TEMPORARILY OFF LINE UNTIL FURTHER NOTICE	Message of the day
.	System ready!
.*LOGIN 115103/320571*)	Sign-On command
JOB 23 PITT DEC-1099/B 63A.41B TTY43 Wed 7-May-80 1815	
Password: (*Your password*))	Supply valid password
Last login: 7-May-80 1619	
Usage ratio: 22.13 Units used: 33.5	
.	System ready!

1.9 Password

To sign on the DEC-10 system, the required identifications are a valid PPN and the associated password. Security of PPNs is impossible because they are publicly displayed in many places - in LOGIN printout, in the file directory, in printout identification, etc. Thus the only real safeguard and security of a computer account is the password.

The need for protection against unauthorized use of your account by another person goes beyond accounting reasons. There have been numerous incidents of computer vandalism in the past. The most frequent vandalism was change or erasure of programs or data without the owner's knowledge.

The only protection against such unauthorized use is to install a password, to keep its security, and to change it frequently. As a matter of prudence and necessity, the user should change his password regularly as a standard practice and whenever he suspects the password is no longer secure.

Changing a password at a terminal can only be done at the LOGIN time by using either of the following LOGIN format:

LOGIN m,n/PASSWORD

or,

LOGIN m/n/PASSWORD

where "m" and "n" are the PPN. The following shows a sign-on session with a password change. Since the process is interactive, the explanation should be self-evident:

```
.LOGIN 115103/320571/password )
JOB 16 PITT DEC-1099/B 63A.41B TTY43 Wed 9-May-80 2003
Password: (Enter old password) )
New Password: (Enter new password) )

Retype for verification

New Password: (Enter new password again) )
Last password update: 24-Apr-80 1255
Last login: 22-Apr-80 1642
Usage ratio: 0.84 Units used: 33.1
```

1.10 Disk Storage Quota

One of the special features of a time-sharing computing system, as compared with a computer for batch processing applications only, is its very large capacity for on-line mass storage, such as the disk storage. It is a common practice to assign and allocate a part of that mass storage for users to store their programs, data or other files. These storage spaces are measured in "disk blocks", or simply "blocks". In DEC-10 system, each block contains 128 data words in DEC-10 format. Therefore, each block can hold a maximum of 640 characters, an equivalent of 8 fully punched cards.

Each authorized user is assigned a quota of disk space in blocks called logout quota, in which he may store his files permanently. These files will not be removed from the storage unless any one of the following situation occurs: (1) when a file is deleted by the user himself, (2) when a file is inactive and not accessed for more than a prescribed period (for example, a month), or (3) when the project has been cancelled or terminated.

When a user is LOGINed and on-line, the actual disk space assigned to him is five times the logout quota. The extra storage is assigned for storing temporary data, non-permanent program or data files needed for the execution of the user's work while he is on line. This on-line quota of disk space is called the login quota. The actual number of blocks assigned as the login quota depends on the logout quota and the available system capacity at the time.

After a user has LOGINed, he may enjoy the larger login quota for his on-line work. When he is ready to sign-off, he must make sure that his disk usage is under the logout quota, otherwise all efforts of signing off would fail, or else the computer will delete the stored files according to a predetermined order of priority until the logout quota requirement is met. In the latter case, the computer may very well delete some important files.

The monitor commands for managing the files are discussed in Chapter 8. However, several commands that are necessary in managing the quota will be briefly discussed here. For more details of these commands, the readers are referred to Chapter 8.

The monitor command R QUOLST is used to inquire about the current status of the disk quota (login, logout, and system status). An example follows. Again, lines in *italics* are typed by the user:

```
.R QUOLST ↵                                    Explanation

User:   115103,320571 ──────────────  User's PPN
Str      used   left:(in) (out)   (sys)
USRB:     180      120     -120    182616   Disk Status:
                                             └── System status
Timesharing Core Class: 0 ──────────────────  Logout quota status
Batch Core Class: 0 ────────────────────────  Login quota status
                  └──────────────────────────  Disk block used
      └──User's core classes
  └──────Storage device specification
```

In this example, the user has a logout quota of 60 blocks and a login quota of 300 blocks. At the time of this inquiry, he has used up 180 blocks. Therefore, the above printout indicates that he is still 120 blocks under the login quota, but he is 120 blocks above the logout quota. Should he wish to sign off at this time, he must first delete his files for at least a total of 120 blocks. So, at this point it is important to him to know how to find out what he has in the storage and how he can selectively delete them. Two other monitor commands useful for quota management are:

```
DIRECT ↵
```

and

```
DIRECT name.ext ↵
```

When the command DIRECT (for "directory") is given, the terminal will print out a list of user's files in the disk storage, along with their names,

extensions, file sizes in blocks and other pertinent information. The total amount of storage occupied is printed out at the end of the list. A sample result of this command is shown below:

```
.DIRECT

TEST    DAT     60  <057>   18-MAY-79    USRB: [115103,320571]
SAMPLE  FOR     48  <157>   19-MAY-79
SAMPLE  REL     36  <057>   22-MAY-79
TEST    BAK     36  <057>   24-MAY-79
  TOTAL OF 180 BLOCKS IN 4 FILES ON USRB: [115103,320571]
```

The command DIRECT thus gives the user an inventory of files in the disk storage at that time. If he is then ready to sign off from the computer, and if he is over the logout quota, this inventory information will enable him to decide which file he should erase in order to get below the logout quota limit. The monitor command of DELETE is used to erase a file in the storage. If in the above example, the files TEST.DAT, SAMPLE.REL and TEST.BAK are to be erased, then the command issued is :

```
.DELETE TEST.DAT, SAMPLE.REL, TEST.BAK
```

After erasure is completed, the terminal will report the names of the erased files and the size of total restored storage. The details of file names, extensions and other information about file name structure are given in Section 1.12.

1.11 Sign-Off Procedure

To leave the system, the user must terminate his job by supplying a monitor command KJOB ("to kill the job"). The system will respond by requesting a code letter for confirmation and file disposition. Thus, the command format for signing-off is:

```
.KJOB ↵
CONFIRM: (code letter)↵
```

A shortened form of this command is:

```
.K/(code letter) ↵
```

The most commonly used code letters in the KJOB command are:

F = fast signoff; save all files

D = fast signoff; delete all files. Computer will respond with A confirming question: "DELETE ALL FILES?" Answer YES and return the carriage.

P = preserve all files except temporary files.

H = HELP! Computer will respond will detailed instructions.

I = list file names, one at a time, and apply code letter decision individually. The code letters for individual decision are:

P = preserve the file

S = save the file

K = delete the file

Q = learn if over logout quota on this file

E = skip to next file and save this file if below logout quota for this file. If not below logout quota, a message is typed and the same file name is repeated.

H = HELP. Computer will respond with the above information on code letters.

While files are disposed per user's code letter instruction, the computer will make a check on logout quota, gather all usage and accounting information, terminate the user's job and print out a summary of the job. For example:

```
.K/F ↵
JOB 16 [115103,320571] off TTY43  at 2032 9-May-80  Connect=29 Min
Disk R+W=83+76  Tape IO=0  Saved all files (450 blocks)
CPU 0:04  Core HWM=11P  Units=0.1263 ($9.48)
```

The printout indicates that this user, with PPN of 115103,320571, was assigned line 43 and job 16, signed off at 2032 on May 9, 1980. His terminal was connected to the system for 29 minutes, used CPU or computer time for 4 seconds. He used disk, but not magnetic tapes. He has 450 blocks of saved files. For this job, the highest core area used (HWM=High-Water-Mark) was 11 pages or 5.5K words, and the charge is 0.1263 unit or $9.48.

The "unit" is an accounting device which combines all charges of the service, including CPU time, disk usage, the length of connect time, the size of core used, and time of the day, and a base charge, each with an appropriate weighting factor to form an accounting formula.

FILES

1.12 Basic Concept of Files

One of the important and convenient features of a time-sharing system is that it is supported by mass storage devices. The need for mass storage during the early days of time-sharing is derived from the fact that only the most important service programs and the program being executed at the moment may be stored in the high-cost, high-speed magnetic core storage. The mass storage serves as a temporary storage for programs and data not being processed at the time. When the user's turn comes, his program and data will enter the core storage. When his allotted time is finished, the program and data in the core return to the mass storage. Such transfer of program and data is an important and unique operation in all time-sharing systems, and is called swapping. The portion of the mass storage, magnetic disk and/or magnetic drum, assigned for swapping is called a swapping device.

The space required for swapping is a relatively small portion of the storage available in the mass storage devices. Thus a time-sharing system generally is characterized by a very high reserve capacity of auxiliary storage. The most frequent use of this capacity is to accommodate users' programs and/or data. These stored programs and/or data are called files.

Each language processor in the time-sharing system contains facilities for file management and file manipulation, and this information will be discussed in various chapters in this book. It will be useful at this point, however, to introduce some basic information and concepts.

The basic unit of information in a file is called a record. If a file is visualized as consisting of a deck of punched cards, then each card becomes one record. The information content of one record varies from case to case. A blank card contains no information, and it is called a null record. A FORTRAN source program record is limited to a maximum of 72 characters/record. For a PIL program, there is no practical limit to the length of a record.

For the purpose of identification, each file is given a name. Once the names are established, the computer will maintain a directory so that users need not be concerned with the exact locations or addresses on the disk to locate their files. For the DEC System-10, the format of a complete name of a file is:

DEV: NAME.EXT [m,n] <xyz>

where:

DEV: = name of device on which the file is stored. If this part is omitted in the complete name, it is understood that the device is user's assigned disk area.

NAME = file name consisting of one to six letters and/or digits, with no embedded blank.

.EXT = file extension consisting of zero to 3 letters and/or digits with no embedded blank. See more explanations below.

[m,n] = the PPN of the person who created or owned the file. Note the use of square brackets.

<xyz> = a three-digit (octal) protection code. See more explanation below. Note the use of angular brackets.

The file extension is the part of file identification used to indicate the language or format of the file. The following are the most frequently used file extensions.

.PIL	A PIL (language) program file
.FOR	A FORTRAN source program file
.REL	A relocatable binary file, or the "object deck"
.BAS	A BASIC (language) source program file
.BAK	A backup file
.DAT	A data file
.TMP	A temporary file
	A null extension (no extension)

Examples:

NEWTON.PIL	A PIL program file named NEWTON.
NEWTON.FOR	A FORTRAN program file named NEWTON.
NEWTON.REL	An object program compiled from NEWTON.FOR
NEWTON.BAS	A BASIC program file named NEWTON.
FOR01.DAT	A data file named FOR01.

Symbols "*" and "?" are used as "wild cards" to represent a class of file names or extensions. The following examples will demonstrate their use:

Examples:

NEWTON.*	All files named NEWTON of any extension.
*.FOR	All FORTRAN files.
.	All files.
F????.DAT	All data files whose names are 5 characters or less and begin with F.
D12??.D??	A files whose names begin with "D12" and contain 5 characters or less, and whose extensions begin with the letter D and contain 3 or less characters.
D12??.*	All files whose names begin with "D12" and contain 5 characters or less.

The protection code is a 3-digit octal number xyz, each digit ranging from 0 to 7. Each digit defines a protection level of the file against a certain class of users:

x = protection level against the file owner himself.

y = protection level against users sharing the same project number.

z = protection level against the general public.

The levels of protection range from 0 to 7, and level 7 is the highest. The exact definition of each protection level is given below:

Code Digit	Access Protection*
7	No access privileges
6	Execute only
5	Level 6 + read privilege
4	Level 5 + append privilege
3	Level 4 + update privilege
2	Level 3 + write privilege
1	Level 2 + rename privilege
0	Level 1 + change protection privilege

The access protection can be changed by executing the RENAME or PROTECT monitor command (see Chapter 8) or by using the service program PIP (see Chapter 7). Since there are 8 levels of protection in each of three classes of users, there are 512 different shades of protection-level combinations possible. Normally, one need only be concerned with a few commonly used codes:

Protection Codes	Applications
077,177	Strictly private and non-sharable, such as grade files maintained by an instructor.
057,177	Sharable within a project, for example, a program to be shared by all students in a course.
055,155	Sharable with the computer community, but the file may not be modified by anyone except the file owner.

The System assigns a default protection level of 057, set automatically by the computer if the person does not specify any protection code when he creates the file. In some coursework, instructors may arrange to have the default protection level automatically set at 077. In such a case, the protection code of a student's file is 077 to his classmates, but is 057 to his instructor.

*Subject to minor local variations. For example, at the University of Pittsburgh, access protection designated by the x-digit has been modified slightly.

EXERCISE ON A TIME-SHARING TERMINAL

For a person with no prior experience with using a computer, it is quite natural for him to feel intimidated when he gets on the computer for the first time. Beginners should feel assured by the fact that very little they do can hurt the computer, except if he gets physically violent and abuses the computer equipment. A session on a terminal to become familiar with its function and operation is highly recommended. The following is a recommended exercise.

(1) With a valid PPN and a password, practise sign-on and change-password procedures. Warning: Do not mix up or forget your new password, or else you will not get back on the computer again.

(2) Once signed on, type any gibberish, return the carriage, and watch the error message from the computer. Always wait for the prompt symbol "." to appear, then type in your line. Don't leave a blank or space after the ".", and don't forget to return the carriage at the end of each line.

(3) Copy a file into your own disk for the terminal exercise. For example, to copy a new bulletin of the System, use the following command:

```
.COPY NEWS.DAT=SYS:NEWS
```

Note the period in the first column is already furnished by the computer; you just type in the rest of the line and return the carriage. After this, use the DIRECT command the and R QUOLST command to find out the status and quota of your disk storage.

(4) After the file is copied into your storage, do the following exercises:

a. Print out the file by the command:

```
.TYPE NEWS.DAT
```

After a few lines are typed out, kill the typing job by either a CTRL-O or multiple CTRL-C (twice or more). The news file is quite long and a complete typeout will take a long time. If you are curious about what the rest of the news bulletin is, apply the following command:

```
.PRINT NEWS.DAT
```

and a printer copy will be produced at the printer. The printout will have your programmer number printed in big block letters on the first page for identification.

b. There is a group of monitor commands that controls the functions of a terminal. They are discussed in Chapter 8 on Operating System commands. However, several commands may be useful enough to the beginner that they will be given here for exercise:

```
.TTY WIDTH n
```

This command will set the right margin of the terminal at the nth column. The value "n" may range from 17 to 200. When you sign on

to the System, the right margin is automatically set at 72.

.TTY PAGE

After this command is given, a CTRL-S will suspend the output (but not kill it), and CTRL-Q will resume it. The purpose is to stop the output in order to examine the output that has already been produced.

After setting the right margin at a new value and giving the TTY PAGE command, repeat the exercise of typing out NEWS.DAT. Use both CTRL-S and CTRL-Q to control the printing.

(5) While still signed on, try to change to another system. Can you do it?

(6) Check your logout quota status. If you are still under your quota, keep "stuffing" your storage by repeating step 3 above (each time using a new file name), until you have gone over the quota. Confirm that by using the R QUOLST and DIRECT commands. Try to sign off in this condition.

(7) Clean up your disk storage and sign off.

(8) Repeat steps 1 through 7 by first signing on purposely on the wrong system. What are the consequences? What is the warning message from the computer? What are the things you cannot do in the wrong system? What are the things you can do in either system?

When you complete this exercise with reasonable facility, you may consider yourself granted a beginner's driver license. Congratulations!

REFERENCES

1. A PRIMER FOR PITT TIME-SHARING SYSTEM (PTSS), T. W. Sze, University of Pittsburgh, Pittsburgh, Pennsylvania; 1970.

2. INTRODUCTION TO A TIME-SHARING SYSTEM, T. W. Sze, University of Pittsburgh, Pittsburgh, Pennsylvania; 1972.

3. ALL ABOUT TELEPRINTER TERMINALS, Datapro Research Coporation, Delran, New Jersey; 1976.

4. LA36 DECwriter II USERS MANUAL, Digital Equipment Corporation, Maynard, Massachusetts; 1974.

5. INTRODUCTION TO COMPUTING AT PITT, DEC-10 Documentation-1, Computer Center, University of Pittsburgh, Pittsburgh, Pennsylvania; April, 1980.

6. UNIVERSITY COMPUTER CENTER, ACADEMIC SERVICES, Computer Center, University of Pittsburgh, Pittsburgh, Pennsylvania; 1978.

7. INDEX OF COMPUTER CENTER DOCUMENTATION AND SERVICES, Computer Center, University of Pittsburgh, Pittsburgh, Pennsylvania; September, 1978.

8. INTRODUCTION TO DECSYSTEM-10: TIME-SHARING AND BATCH, T. W. Sze, University of Pittsburgh, Pittsburgh, Pennsylvania; First Edition, 1974; Second edition, 1977.

CHAPTER 2

TEXT EDITOR

Everything must have a beginning. From a user's point of view, his starting point is to enter his program and/or data into the computer. The DEC System-10 is mainly disk-based machine. That means, the computer will look in the user's disk area for the program a user wants to execute. Therefore, in order to do any computer processing, a user must first place his program and/or data in the disk. The text editor is a system program that will enable the user to perform this task.

2.1 Introduction

The UPDATE (University of Pittsburgh DAta and Text Editor)* is a service program with which a user can correct, modify, duplicate, or delete parts of a stored program or data file.

In order to edit a program or data file, it must be one already stored on disk, magnetic tape, or DECtape. However, if the source material is on tape, using UPDATE will result in an edited copy of material stored on disk, and the original material on tape is unchanged. Ultimate changes on tape still require the use of another service program, such as PIP (see Chapter 7), to delete the old file on tape and to transfer the new file from disk to tape. Otherwise, the new file is re-copied onto the tape so that the old and the new copies co-exist together on the tape. Therefore, for all practical purposes, UPDATE is used as a disk-to-disk editor, taking source material from the disk and storing the edited copy back on the disk. Discussions in this chapter are based on such disk-to-disk editing.

After the user signs on in the usual way, he can get the service of UPDATE by typing the following monitor command:

```
.R UPDATE
```

or

```
.UPDATE
```

*Developed by Gerald W. Bradley, University of Pittsburgh (Reference 7).

When the UPDATE editor is assigned, the computer will first ask for the name of the input file to be edited in this manner:

```
.R UPDATE
INPUT=>
```

The user will then type in after the greater-than sign the file name, extension, file owner's PPN if the user is not the owner. For example:

```
.R UPDATE                    .R UPDATE
INPUT=>SAMPLE.FOR            INPUT=>SAMPLE.FOR[115103,320571]
>                            >
```

When the greater-than sign is again printed by the computer, UPDATE is ready to accept editing commands, and editing on the specified file can begin. As the editing proceeds, whenever UPDATE is ready to accept a command or an insertion, a sign ">" is printed out as a prompt symbol. The first space after the sign should be considered as column No. 1.

The above process may be shortened by using the following formats:

```
.UPDATE SAMPLE.FOR              .UPDATE SAMPLE.FOR[115103,320571]
>                               >
```

2.2 Selected Terminology

The following terms will be used quite frequently in the discussion of the UPDATE commands:

(1) Record, or Line

A record or a line is a basic unit of information in a file. If a file consists of a deck of punched cards, each card becomes one record. For a file stored on disk, one record actually is a tiny length of track on the disk. The information content for one record varies from case to case. In a FORTRAN program, each record is limited to a maximum of 72 characters including blanks, although each statement may extend for several records if needed. Sometimes, there is no information at all on a record, such as a blank card, and this is called a null record.

(2) Pointer

Once the input file is specified and loaded, the UPDATE at that time is positioned at the first record, or line, of the file. At that point, editing commands will refer to text material with that line as a reference point. Later, if one wishes to make editing steps at another line, the UPDATE should be re-positioned by appropriate commands. For convenience, we shall assume an imaginery "pointer" which indicates the position of the record being aligned. Thus, such a statement as "moving the pointer forward 5 records" should now make sense.

(3) Line Numbers, Absolute and Relative

A file begins with record No. 1, then No. 2, etc. Such line numbers represent the true positions of the records in the file, and are called the absolute line numbers. On the other hand, it is often convenient to use as a

reference the line currently pointed to and say, for example, "move forward 5 lines" or "back up 3 lines". These are then relative line numbers. Absolute line numbers are always expressed by unsigned positive integers, and relative line numbers by signed integers. Use "+" sign for forward reference and "-" for backward reference in specifying relative line numbers. Note that a file always begins at line number 1, and its line numbers are always contiguous. Therefore, if lines 4 and 5 are deleted during editing, then line 6 becomes line 4, 7 becomes 5, etc.

(4) Delimiter While the pointer indicates the position of a line in a text, the position of text within a line is indicated by the use of delimiters. These delimiters may be thought of as quotation marks in the English language, except that any special character may be used as a delimiter in UPDATE. Thus, if one wishes to set off the last three words of this particular paragraph, he may specify:

"he may specify:" or /he may specify:/
or ?he may specify:? or $he may specify:$ etc.

Because of its similarity with the quotation, the string set off by a pair of delimiters will be referred to as a "quoted string" or simply a "quotation". There are several important rules of delimiter usage in the UPDATE editor:

A. Use consistent characters as delimiters for a quotation. While any special character may be used as a delimiter, the choice of the beginning-of-quotation (BOQ) delimiter automatically decides the use of the same character as the end-of-quotation (EOQ) delimiter. The following examples should be self-explanatory:

Valid Use of Delimiters	Invalid Use
"quoted text"	(quoted text)
(quoted text(	<quoted text>

B. If a quoted string contains a special character, that particular character should not be used as a delimiter for this quotation. For example, if we wish to quote a string "less than $5.00" and use "$" as a delimiter, the result will be misinterpreted by UPDATE.

C. If several quotations are placed in one UPDATE command, the following rules apply:

   a. The first BOQ delimiter determines the character to use.

   b. Multiple quotations must all refer to materials on the same record.

   c. When multiple quotations are placed together, two adjacent delimiters should always be merged into a single one to avoid ambiguity. In other words, a delimiter should not only serve as the BOQ delimiter for the following quotation, but also as the EOQ delimiter for the preceding quotation. Thus a general appearance of a multiple quotation will be something like this:

/QUOTE 1/QUOTE 2/QUOTE 3/

If this multiple quote is written as:

/QUOTE 1//QUOTE 2//QUOTE 3/

it will actually be interpreted by UPDATE as 5 quotations, the second and the fourth being null strings.

D. The contents of a quotation must be exact and unique. When UPDATE receives a quoted string, it will try to search in the pointed line for a group of characters exactly matching the quotation. In such a matching process, the capital letters, the lower cases, the blanks, special characters, and control characters are all legitimate and different characters. For this reason, a quoted string must be given in the exact way as in the pointed line.

Example: Suppose we wish to quote the underscored portion below:

50 Y1 = Y0 + Y 1

Correct Quotation	Incorrect Quotation
/Y 1/	/Y1/

E. When UPDATE searches a line text to match a quotation, it begins with the character in column one. As the search moves to the right, and a match is found, the search is completed. If a quoted string appears several times in a text line, UPDATE will always pick the string nearest to the first column. Therefore, if we wish to sepcify non-unique strings further to the right, the string must be expanded in front and/or in the back until the string is unique, or else it is the first such quotation when the search starts from the left end.

Example: Suppose we wish to quote the underscored portion below:

501 IF(Y.LT.0.0001) GO TO 510

Correct Quotation	Incorrect Quotation
/0.0001/	/0/
/.000/	/01/
/01)/ etc.	/00/

F. A single quotation followed immediately by an integer means this quotation begins from column indicated. For example, the quotation /01/19 means the character string "01" that begins at column No. 19.

G. All quotations must be bracketed within a pair of delimiters. Unclosed quotation is an error.

A PRIMER OF UPDATE EDITOR

The text editor UPDATE contains several dozens of editing commands. For a beginner, it would be a mistake to attempt to learn them all at one time. Experience has shown that most editings are done with a limited set of editing commands. Complex command functions can usually be accomplished by applying several simpler commands in sequence. For the sake of learning efficiency, it would be much more cost effective for a beginner to concentrate on a few basic editing functions and commands. They are:

(1) To move a pointer to a designated line.

(2) To make changes on a pointed line.

(3) To delete the pointed line or lines.

(4) To type out the content of the pointed line or lines.

(5) To insert a line at a designated place.

(6) To conclude the editing.

The commands given in the following sections pertain to these basic functions. All UPDATE commands must have a "$" in the first column. The spelling of each command may be shortened to just the first two letters. Misspelling after the first two letters will be ignored and will not be considered an error.

2.3 Movement of Pointer, $TO, $AT and $TRAVEL

When the UPDATE first opens an input file, the pointer is always positioned at line No. 1. There are three commands one may use to move the pointer elsewhere, and they are $TO, $AT and $TRAVEL.

$TO will move the pointer to a specified place, and once there the new line is typed out for verification.

$AT performs the same function as $TO, but the typing of a new line is suppressed.

$TRAVEL performs the same function as $TO, and the command is "remembered". The same $TRAVEL command can be executed again later by issuing a $GO command.

All three commands have the same command formats and variations, and those for $TO are listed below. Variations of formats would be the same for the other two commands, simply by replacing $TO in the following listing by either $AT or $TR.

A.	$TO N	Move the pointer to line N.
B.	$TO +N	Move the pointer N lines forward.
C.	$TO -N	Move the pointer N lines backward.
D.	$TO /TEXT/	Move the pointer forward from the present line until it encounters a line with the exact string /TEXT/ in it.
E.	$TO $TEXT$	Similar to case D above, with an exception that the match will not consider the difference between upper and lower case letters, nor will it take into account any blank between characters.
F.	$TO /TEXT/K	Search for the string TEXT that begins at column No. K.
G.	$TO $	Move the pointer to the last line.

Notice the mode of search in cases D, E and F. The search starts from the line below the pointed line. If there is a string TEXT in the pointed line, it will not be found. If one wishes to move to the first appearance of /TEXT/ or $TEXT$ while he is in the middle section of the file, he should issue the command $TO 1 first before giving a $TO/TEXT/, or $TO/TEXT/K, or $TO $TEXT$$ command. This is so that he will not miss any earlier existences of the string TEXT in the lines before the pointer. But even so, such a search would miss line 1, unless the user examines the line typed out after the command $TO 1. This problem may be solved by inserting a blank line as line 1 and later removing it before finishing the editing job.

To move forward one line, two ways are possible: Either use $TO +1 command, or simply return the carriage.

To move backward one line, either use the command $TO -1, or press backspace key and then return the carriage.

While moving the pointer back and forth during the editing job, it will become difficult to keep track of the line number of the pointer. The command $WHERE will cause a number typed out enclosed in brackets to indicate the current pointer line number.

Example: Suppose you wish to examine every FORMAT statement in your FORTRAN program. You will first call the file using the UPDATE. Then apply the following command:

```
>$TRAVEL /FORMAT/
```

The first time you apply it this way, it will move the pointer to the first FORMAT statement, and print it out. Any revision of the statement may be done there and then. The movement to the subsequent FORMAT statements may be accomplished by giving the command:

```
>$GO
```

Naturally, if a FORMAT statement in the program is misspelled, the $TRAVEL command will not find that statement.

2.4 Change of Text Material, $CHANGE, $ALTER and $SUBSTITUTE

When the appropriate line is positioned by the $TO, $AT or $TRAVEL command, editing changes may be performed using the $CHANGE, or $ALTER command. The standard format is:

$CHANGE /OLD TEXT/NEW TEXT/

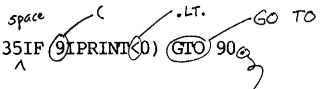
For multiple changes on the same line, the command format is:

$CHANGE /OLD 1/NEW 1/OLD 2/NEW 2/OLD 3/NEW 3/...

The rules of delimiters in multiple quotations have been discussed before and are applicable here. Again, the delimiters for multiple quotations must be consistent for all quotations.

As a convenience feature, after the $CHANGE command is executed, the entire new line is typed out for verification. The pointer position remains unchanged.

Example: Suppose the indicated changes are required as shown below:



The following two lines show first the editing command of $CHANGE and then the edited text automatically typed out after execution. (Remember our convention in this book --- User's input line shown in *italics*):

>$CHANGE/5/5 /9/(/>/.LT./G/GO /.//
35 IF (IPRINT.LT.0) GO TO 90

There are, of course, many other ways to write the above $CHANGE command to achieve the same result.

$ALTER is used in the same way as the $CHANGE command, except that $ALTER does not allow multiple changes. Its main usefulness is in the compounded editing commands, as will be illustrated in a later section.

$SUBSTITUTE differs from $CHANGE or $ALTER in this manner: The command $CHANGE or $ALTER is used to change a string in one single line positioned by the pointer. The command $SUBSTITUTE is used to alter a string in the entire file beginning from the pointed line. Again, the string of characters to be changed must be specified exactly and uniquely. Otherwise, inadvertent changes will result at unintended places. For example, if one wishes to change the variable X into Y in a certain program, specifying $SUBSTITUTE/X/Y/ would change every X-character into Y-character. Thus, inadvertently, another variable with the name "INDEX" would become "INDEY", and the exponential function name EXP would be changed to EYP.

There are two variations for the command $SUBSTITUTE:

A. $SUBSTITUTE /OLDTEXT/NEWTEXT/

Starting from the pointed line, this command will search for a string /OLDTEXT/ and each time upon finding it, change it into /NEWTEXT/ until the end of the file is reached.

B. $SUBSTITUTE /OLDTEXT/NEWTEXT/K

Starting from the pointed line, this command will search for a string /OLDTEXT/ that begins at the Kth column, and each time upon finding it change it to /NEWTEXT/ until the end of the file is reached. After the $SUBSTITUTE command is executed successfully, the pointer will be relocated at the last line of the file.

2.5 Deletion of Lines, $DELETE

When a line or a group of consecutive lines are to be deleted, first position the pointer to that line or the first line of that group, using either the $TO or $AT command. Then depending on what is to be deleted, use the command $DELETE in the following ways:

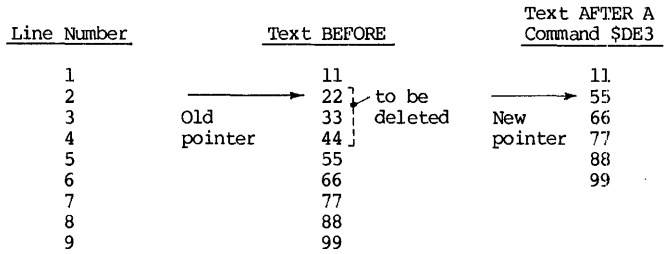
(1) $DELETE N Delete N lines beginning with the one presently pointed to. After the deletion, the pointer moves forward to the line immediately after the deleted group. If the deleted group happens to be the final N lines of the file, the pointer drops back one line and positions at the new last line. If N is larger than the number of lines left on the input file, a command $DELETE N will delete every line remaining and then type out a "?" to indicate error. This feature is actually quite useful when one wishes to delete the rest of the text but does not know how many lines there are. Then, he can simply issue a command of $DELETE 10000, or any number larger than the number of remaining lines.

(2) $DELETE This is automatically interpreted as $DELETE 1.

(3) $DELETE $ Beginning with the currently pointed line, this command will erase the rest of the file.

$DELETE -N is NOT a valid command.

Example: See below for the "Before" and "After" with a $DELETE command:



Note that although the pointer is positioned at a new line of text, the line number of the pointed line remains the same. The line numbers and the text are then automatically readjusted.

2.6 Output of Lines, $TYPE

Frequently, it is desirable to display the text of a line on the terminal for examination. The UPDATE command for this function is $TYPE. The following shows the variations:

A. $TYPE N — This command will type out N lines beginning with the present line. The position of the pointer remains unchanged.

B. $TYPE — Same as $TYPE 1.

C. $TYPE $ — This command will type out the currently pointed line and the last line of the file. Pointer position remains unchanged.

2.7 Line Insertion

UPDATE will regard any user input as UPDATE command if column 1 is a "$" character. Conversely, UPDATE will regard any user input as line insertion if the line does not begin with a "$" in column 1. When a line is inserted, it will always be inserted after the currently pointed line. If you wish to insert a line before the pointed line, you must precede your insertion by a "$BEFORE" command. When a new line has been inserted, the pointer will move forward one line, making the new insertion the currently pointed line.

Beside adding lines to an old file, this process is particularly useful in creating new files. The process of creating a new file is outlined as follows:

(1) Call for UPDATE and give a file name that does not yet exist. For example:

```
.UPDATE NEW.FOR
```

where NEW.FOR is a file name given to the new file to be created.

(2) The pointer of the blank file called by the UPDATE will be positioned at line zero. Type in the new file, one line at a time. Each line is terminated by a carriage return in the conventional way of typing.

(3) While the new file is being created, the editing commands can be applied to move the pointer, to type out the lines, to delete or to change the contents of a line.

(4) When all lines are entered, exit from UPDATE by a command $END.

2.8 Completion of an Editing Session, $DONE, $END and $FINISH

All three commands signify the end of current editing of the file. They differ in how the file should be stored and named.

When the $DONE command is issued, UPDATE will ask the user to supply a file name for the edited file, for example:

```
>$DONE
CATALOG NAME=>SAMPLE.FOR
6 BLOCKS WRITTEN ON SAMPLE.FOR[115103,320571]

EXIT

.
```

If the file name and the extension given here are exactly the same as those of the old file, the old file is replaced by the new file. As a safety measure, the old file is retained in the storage with the extension changed to BAK (for "backup"), in case the user changes his mind about his revisions. If either the name or the extension or both are different from those of the old file, a new file is created and stored on disk along with the old file, and the old file is not disturbed. If the name and the extension given during the cataloging are exactly the same with those of some other file in the disk storage, naming two different files with the same name causes an error, and the UPDATE will reject the duplicate name and ask for a new name. This is illustrated below:

```
>$DONE
CATALOG NAME=>NEW.FOR
FILE DSK:E20016.TMP[115103,320571]to NEW.FOR[115103,320571]
        RENAME error (4) - Already existing file
CATALOG NAME=>SAMPLE.FOR
6 BLOCKS WRITTEN ON SAMPLE.FOR[115103,320571]

EXIT

.
```

The catalog name can also contain a protection code specification, for example, SAMPLE.FOR<155>. When the protection code is omitted, the UPDATE will automatically assign a protection code of 057.

When the $END command is issued, a fast exit is accomplished and the edited file will have the same file name, extension and protection code as those of the old file. The old file becomes a BAK file. After the storage process is completed, UPDATE returns the user to the monitor.

When the $FINISH command is issued, it will perform the same function as $END. However, instead of returning the control to the monitor, the user will retain the service of the UPDATE editor and can then start a new editing job. Therefore, this command is equivalent to issuing two successive commands: an UPDATE command of $END followed by a monitor command of .R UPDATE.

OTHER UPDATE COMMANDS AND PROCEDURES

When UPDATE is called, several events happen:

(1) The UPDATE program is loaded into the computer memory assigned to the user.

(2) Two disk areas are assigned as working files. One is used as the input file labeled as E1 and the other is used as the output file labeled as E2. The actual file names assigned are E100xx.TMP and E200yy.TMP respectively, where "xx" and "yy" are numbers arbitrarily assigned.

(3) After the input file name is given by the user, as requested by the UPDATE, a copy of that file is loaded into E1. If no such file name exists, E1 remains a blank file. In either case, E2 is a blank file at this point.

(4) UPDATE will read up to 100 lines (which may be specified and modified by a $FACTOR command) from E1 file into the memory.

The logic flow of the text information during an editing session is shown in Figure 2.1.

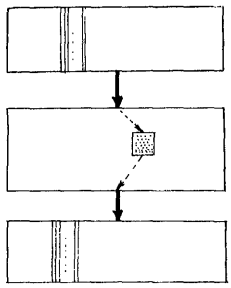


Figure 2.1
Editing Input/Output Files

Now, as the editing session progresses and the pointer advances through the file, more lines are read into the memory. When the number of lines in the memory is more than 100, or whatever value specified by a previous $FACTOR command, the lines behind the pointer are written into the E2 file. Thus, if the pointer keeps advancing forward, more lines are transferred into E2. When the editing is finally completed, all lines in the core, and all the remaining lines in the E1 file are copied onto the E2 file. The E2 file is then renamed by a name designated by the user, and the E1 file is erased. It is significant to note from Figure 2.1 that the movements of lines from E1 to the computer memory, then onto the E2 file, is always in one direction only.

Thus, if the pointer is moved backward, there will be complications. If the pointer, after moved backward, is pointing to a line still in the core memory, events are still normal. If the pointer is positioned at a line no longer in the core memory, that line has already been copied onto the E2 file and cannot be retrieved because the transfer between the memory and E2 is one-way only, as shown in Figure 2.1. This will set forth a sequence of events described as follows:

First, the lines in the memory and all the remaining lines in the E1 file will be copied onto the E2 file. The E2 file is then closed. The E1 file is erased. The E2 file is renamed as the E1 file. The new E1 file is read into the computer memory containing the line positioned by the pointer. The backing-up of the pointer is now finally accomplished. These events resemble a

situation when a driver misses an exit on a one-way urban beltway. In order to exit at the missed exit point, he must drive the whole way around the one-way highway and gets off at the desired exit. However, such events at the editing session are "transparent" to the user, because at the terminal he will be unaware of these. But this situation does suggest that backing up in positioning a line should be done sparingly.

When E2 is closed, it is renamed by a name designated by the user if the closing command is $DONE, and the input file is not disturbed. If the $END or $FINISH command is used, the E2 file is renamed by the same input name, and the input file is renamed as a BAK file.

If the editing involves an auxiliary file as an input or output for the editing, another disk file labeled E3 is assigned. This happens with the command $ONTO or $FROM (See Section 2.16).

2.9 Line Insertion Mode

UPDATE will treat all input lines that start with a $-sign in the column 1 as an UPDATE command. Conversely, UPDATE will treat any input information without a $-sign in column one as a non-command and as information to be inserted in the text.

There are two modes of line insertion:

(1) Insertion after the pointer

A. Insertion of lines typed at the terminal

Any input information to the UPDATE without a dollar sign in column one will automatically be inserted immediately after the current line. When the insertion of one line is completed, the pointer moves forward one number, so that it is now positioned at the newly inserted line. The next typed line will be inserted immediately after the previously inserted line, and again the pointer moves to the newly created line. This feature makes it very convenient to use the terminal keyboard to create a file.

The insertion mode is suspended whenever an UPDATE command (with a $-sign in the first column) is issued.

Example: Observe the "Before" and the "After" of an insertion procedure:

Line Number	Old Text and Pointer Position	User types in:	New Text and Pointer Position
1	11	*AA*	11
2	→ 22	*BB*	22
3	33	*CC*	AA
4	44		BB
5			→ CC
6			33
7			44

So the UPDATE interprets every input line beginning with a $-sign as an UPDATE command. This may develop into a dilemma if the user attempts to insert a line that begins with a $-sign. For example, consider the statement: "$5.00 IS TOO MUCH TO PAY". When this statement is inserted, UPDATE will puzzle over the meaning of "$5." as an UPDATE command, and the execution results in an error report.

There are several ways to solve this problem: One is to insert a line: "X5.00 IS TOO MUCH TO PAY", and then use $CHANGE command to change the first "X" into "$". Another way is to insert the line: " $5.00 IS TOO MUCH TO PAY", leaving a blank in column 1, and then remove it using the $CHANGE command. If there are many such statements to insert (for example, in preparing a control file for batch processing), the process may be simplified by an UPDATE command:

```
$IS #
```

where "#" can be any special character except ";". The effect of this command is to replace the format of all subsequent editing commands from $XX to #XX, therefore allowing insertion of lines beginning with "$", but disallowing insertion of lines beginning with "#". A command #IS $ later will restore the UPDATE to the normal command format.

To insert a blank line, one should not simply press the carriage return, because that action would merely move the pointer forward one line, and no insertion of any kind is accomplished. A blank line may be inserted by typing (at least) one blank then returning the carriage.

B. Insertion of a stored file

If the lines to be inserted are already stored on disk as a file whose name is given, for example, as NAME.EXT, by its owner with PPN of [m,n], the insertion can be made simply in this manner:

a. Position the pointer at the line immediately before the insertion.

b. Issue the following UPDATE command:

```
$INPUT = NAME.EXT[m,n]
```

As usual, if [m,n] are the user's own numbers, they may be omitted in the command. After the insertion, the pointer moves forward to the last inserted line. This command is frequently used for merging parts of program or data files.

Although the inserted lines come from a stored file on the disk, the UPDATE editor treats them the same way as if they come from the terminal. And hence, the lines in a stored file insertion are subject to the same UPDATE editing rules, particularly about the interpretation of the first column character. If a file will be used as a straight forward insertion of lines, it should be inspected first to see if there is no "$" sign in the first columns. If there is any "$" in the first column, appropriate action, such as "$IS #" command, should be taken prior to the insertion. On the other hand, another avenue of issuing editing commands in addition to the terminal is now opened up. One now may use either the terminal or a stored file to issue editing commands.

UPDATE is greatly enhanced when a sequence of fixed UPDATE commands, which will be executed frequently, is stored as a file. Execution of this sequence of commands can be carried out automatically simply by the $INPUT command. These stored files now become editing programs and can be used over and over.

Example: The following is a file ATTEND.DAT that needs updating each week. The contents with the column numbers are shown below:

```
(Column )                  11111111112222222223
(Numbers )        12345678901234567890123456789 0

                  PERSON A       10110      1
                  PERSON B       10011      2
                  PERSON C       01101      3
                  PERSON D       10111      4
                  . . .
                  PERSON Z       00110    126
```

Suppose the requirement of updating the file is as follows. Remove column-16; shift columns 17-20 to the left by one column; and replace column-20 by zeros.

An "editing program" may be designed and stored as EDIT.PRG that contains the folowing statements:

```
$AT1
$SUBSTITUTE /1/0/16
$AT1
$SUBSTITUTE /0//16
$AT1
$SUBSTITUTE / /0 /20              ( =blank)
```

This sequence may be executed as shown below:

```
.UPATE ATTEND.DAT
>$INPUT=EDIT.PRM
>$END
1 BLOCK WRITTEN ON ATTEND.DAT[115103,320571]

EXIT
```

When this editing program is executed, columns 17-20 will be shifted to the left by one column.

(2) Insertion before the pointer

To insert material before the pointer, first apply the command $BEFORE. Then all lines with no ($) sign at the column-1 will be inserted before the pointer. In the meantime, the pointer will move to the last inserted line. The Insertion mode is terminated by any UPDATE command. See the example below:

Example: Observe the "Before" and "After":

Line Number	Old Text and Pointer Position	User types in:	New Text and Pointer Position
1	→ 11	*$BEFORE*	AA
2	22	*AA*	→ BB
3	33	*BB*	11
4	44		22
5			33
6			44

2.10 Compounded Editing Commands

The UPDATE commands discussed so far have the format of one command per command line. When several commands are issued on a single command line, they become a compounded command. The general format of a compounded command is:

$COMMAND 1; COMMAND 2; COMMAND 3; ...

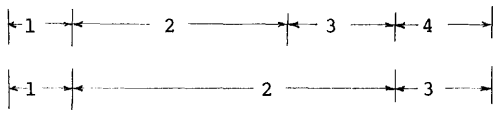
The semicolons ";" are used to separate the successive commands, and therefore no semicolon should appear after the last command in the compounded structure. Also, if any of the commands contains a quotation of string, the string must not contain any semicolon-character, because it will be misunderstood as a command delimiter. Note that the dollar sign "$" is needed only for the first command. There are several straight-forward rules for constructing a compounded UPDATE command:

(1) All commands of a compounded command must fit in a single command line.

(2) The individual commands in the compounded command are executed in their natural order from left to right.

(3) Certain commands may cause ambiguity and error if they are followed by other commands in a compounded structure. Consider the following compounded command:

$TO 5; CHANGE /OLD1/NEW1/; TO/TEXT/; TYPE 4



Interpretation 1:
4 single commands

Interpretation 2:
3 single commands
with multiple string
changes in $CHANGE

It can be seen that the interpretation is ambiguous and it will be unpredictable how this command would be actually executed. To avoid this problem, commands of this kind are always regarded as the last command in the structure, even if there are more commands after them. If more commands are given after them in a compounded command, the added commands are simply ignored, and no error return

signal is returned. Thus, when the above example is executed, the part "TO /TEXT/; TYPE 4" will not be executed. In order to accomplish the function of the above compounded command, the above example should be modified to:

```
$TO 5; ALTER /OLD1/NEW1/; TO /TEXT/; TYPE 4
```

The ambiguity is now removed because the $ALTER command can allow only one change of string.

There are certain UPDATE commands that must be physically the last command in a compounded structure. These commands are listed below:

Group	UPDATE Commands
Multiple string change	CHANGE
Change of command format	IS
Auxiliary file operations	INPUT, ONTO, FROM
End of editing session	END, DONE, FINISH

Commands appended to any of the above commands in a compounded structure will simply be ignored.

Compounded command structure format provides a convenience for input commands. It also is a basis on which a simple and powerful editing program can be built, especially when it couples the usage of $TRAVEL and $GO commands in the compounded structure:

Example: $AT 1; TRAVEL /FORMAT/7; WHERE; GO
Function: Beginning at line 2, search for the string of characters "FORMAT" that begins at column-7. When it is found, type out the line itself and the line number. Repeat the function until the end of the file is reached. In other words, this compounded command will print out all FORMAT statements and where they are in a FORTRAN program. Notice this compounded command will miss line 1; why?

Example: $AT1;TR/ /;AL/ / /;AT-1;GO
Function: Beginning from line 2, all multiple blanks will be reduced to single blank.

Example: $TR/C/1;DE;AT-1;GO
Function: Remove all Comment Lines in a FORTRAN program.

2.11 Move Command, $MOVE

This command will move a block of lines to somewhere else in the file. There are two general formats: One is a single-command format, the other a multiple-command format.

(1) Single command format

The merging of $MOVE N and $TO commands forms a single-command that will move an N-line block to a place designated by the $TO command. Before the move, the pointer should always be positoned at the first line of the N-line block. Immediately after the move, the pointer will always be at the last line of the N-line block at its new place. Because of the $TO command, there are many variations of the $MOVE N TO commands. They are listed below:

A.	$MOVE N TO M	Move N-line block to a new position so that the first line of the block is now line No. M.
B.	$MOVE N TO +M	Move an N-line block to a new position starting immediately after the line which has a relative line number of +M from the last line of the block before the move.
C.	$MOVE N TO -M	Move an N-line block to a new position immediately before the line which has a relative line number of -M, relative to the first line of the block.
D.	$MOVE N $	Move an N-line block to the end of the file.
E.	$MOVE N TO /TEXT/	Move an N-line block and place it immediately after the line beyond the pointer that has the first appearance of the string /TEXT/.
F.	$MOVE N TO /TEXT/K	Move an N-line block and place it immediately after the line beyond the pointer that has the first appearance of the string /TEXT/ that starts at the Kth column.

Example: $MOVE N TO M

Line Number	Old Text and Pointer Position	$MOVE Command	New Text and Pointer Position
1	11	*$MOVE 2 TO 3*	11
2	→ 22		44
3	33		22
4	44 ←		→ 33
5	55		55
6	66		66
7	77		77

Example: $MOVE N TO +M

Line Number	Old Text and Pointer Position	$MOVE Command	New Text and Pointer Position
1	11	*$MO 2 TO +3*	11
2	→ 22		44
3	33		55
4	44		66
5	55		22
6	66		→ 33
7	77		77

Example: $MOVE N TO -M

Line Number	Old Text and Pointer Position	$MOVE Command	New Text and Pointer Position
1	11	*$MO 2 TO -3*	11
2	22		55
3	33		→ 66
4	44		22
5	→ 55		33
6	66		44
7	77		77

Example: $MOVE N TO $

Line Number	Old Text and Pointer Position	$MOVE Command	New Text and Pointer Position
1	11	*$MO 2 TO $*	11
2	→ 22		44
3	33		55
4	44		66
5	55		77
6	66		22
7	77		→ 33

Example: To interchange a pointed line with the next line.

Line Number	Old Text and Pointer Position	$MOVE Command	New Text and Pointer Position
1	11	*$MO 1 TO +1*	11
2	→ 22		33
3	33		→ 22
4	44		44
5	55		55
6	66		66
7	77		77

Example: To interchange a pointed line with its preceding line.

Line Number	Old Text and Pointer Position	$MOVE Command	New Text and Pointer Position
1	11	*$MO 1 TO -1*	11
2	22		→ 33
3	→ 33		22
4	44		44
5	55		55
6	66		66
7	77		77

Example: $MOVE N TO /TEXT/

Line Number	Old Text and Pointer Position	$MOVE Command	New Text and Pointer Position
1	11	*$MO 2 TO '55'*	11
2	22		44
3	33		55
4	44		22
5	55		→ 33
6	66		66
7	77		77

Example: $MOVE N TO /TEXT/K

Line Number	Old Text and Pointer Position	$MOVE Command	New Text and Pointer Position
1	1b	*$MO 2 TO /3b/1*	1b
2	→ b1		b2
3	2b		3b
4	b2		b1
5	3b		→ 2b
6	b3		b3
7	4b		4b

(b=blank)

The search for /TEXT/ starts from the next line from the current line. Thus the search will omit the current line and all lines prior to that. The following example shows an error of search:

Example: $MOVE N TO /TEXT/

Line Number	Old Text and Pointer Position	$MOVE Command	New Text and Pointer Position
1	11	*$MO 2 TO /33/1*	11
2	22		22
3	33		33
4	44		44
5	→ 55]		→ 77
6	66]		
7	77		

(Search unsuccessful when reaching the end of file, and moved lines are lost.)

(2) Multiple command format

Moving an N-line block of text can also be achieved with first a $MOVE N command, and then when the destination is accurately positioned, with another $HERE command. What actually happened is that the N-line block is temporarily stored in an auxiliary file E3, and when the $HERE command is given, the lines will re-enter the computer memory. The advantage of moving lines in this manner is that the procedure becomes less error prone because of accurate positioning of the destination. In the nine examples shown above for the single-command format, movements of lines can also be accomplished by a three-step procedure: $MOVE N, accurate positioning by $TO, and then $HERE commands. Observe the difference in the line numbers used between the single-command and the multiple-command formats.

2.12 COPY Command

This command will duplicate a block of lines elsewhere in the file. The format of the command is very similar to that of $MOVE, and so are the variations. Instead of using TO for positioning in the $MOVE command, $COPY uses AT for positioning the pointer. The variations of $COPY are listed below with similar definitions as applied to the $MOVE variations:

(1) Single command format

A. $COPY N AT M

B. $COPY N AT +M

C. $COPY N AT -M

D. $COPY N AT /TEXT/

E. $COPY N AT /TEXT/K

F. $COPY N AT $

(2) Multiple command format

$COPY N

Accurate positioning command

$HERE

These variations are again illustrated by examples:

Example A: $COPY N AT M

Line Number	Old Text and Pointer Position	$COPY Command	New Text and Pointer Position
1	11	*$CO 2 AT 3*	11
2	→ 22		22
3	33		22
4	44		→ 33
5	55		33
6	66		44
7	77		55
8			66
9			77

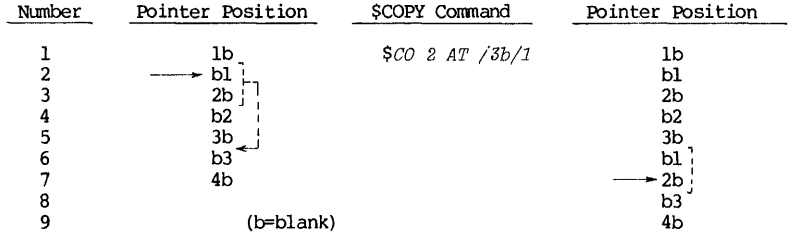
Example B: $COPY N AT +M

Line Number	Old Text and Pointer Position	$COPY Command	New Text and Pointer Position
1	11	*$CO 2 AT +3*	11
2	→ 22		22
3	33		33
4	44		44
5	55		55
6	66		66
7	77		22
8			→ 33
9			77

Example C: $COPY N AT -M

Line Number	Old Text and Pointer Position	$COPY Command	New Text and Pointer Position
1	11	*$CO 2 AT -2*	11
2	22		22
3	33		33
4	44		44
5	→ 55	Watch out for	55 -2 lines
6	66	tricky minus	→ 66
7	77	count here. A	55
8		poor feature.	66
9			77

Example D: $COPY N AT /TEXT/K



The four examples above show how $COPY command may be used in a single-command format. If $COPY is used in a multiple-command format, the commands to produce the same results as the above four examples will be:

Example A	Example B	Example C	Example D
$CO 2	$CO 2	$CO 2	$CO 2
$AT 2	$AT+2	$AT -3	$AT/3b/1
$HE	$HE	$HE	$HE

Since the pointer will be positioned at a line beyond the line of the copied group after each $COPY N command, the counting of lines is different. Therefore, observe particularly the number of lines of movement for the pointer in the first three cases.

2.13 Editing-Control-Function Switch Commands

There is a group of editing control functions that UPDATE can turn them ON or OFF by commands. When a function is switched ON, that function will be in force. Such software switches have many similar properties as a hardware switch. For example, a function will be OFF unless explicitly turned OFF, or vice versa. Turning ON a switch several times in succession is equivalent to turn it on just once.

(1) Functions permanently switched ON or OFF by UPDATE

The following is a group of editing commands that provides a variety of control functions during an editing session. It has a general format of

```
$KEYWORD = YES
$KEYWORD = NO
```

where KEYWORD represents an option, and YES or NO to indicate whether such option is to be switched ON or OFF. When a function is switched ON, the effect is permanent for the remainder of the editing session or until the function is explicitly turned OFF. When UPDATE is first called, all these switches are in the OFF condition.

A. $ARROW=YES; $ARROW=NO When this option is turned on, all control characters in the current line can be displaced by the TYPE command as either ^-character or ^-character, such as ^L or ^L, ^I or ^I.

Example: Often, in entering a text line, the shift key of the terminal is used to enter special symbols, such as "*" or "]". If by mistake, the control key is used instead of the shift key, the mistake cannot be easily detected because a control character will not be echo-printed. Observe the following segment of an editing session:

UPDATE Commands	Comments
>$TYPE	
DIMENSION X(10)	Printout seems OK
>$ARROW=YES	
>$TYPE	
DIMENS^[ION	Hidden non-print character
>$CHANGE /^[//	Remove it.
DIMENSION X(10)	It's gone.
>	

B. $EDIT=YES; $EDIT=NO When a $EDIT=YES command is given, the UPDATE automatically inputs and prints out a "$" sign in column one. The user can thus enter the command keyword directly without the "$" sign. Unless there is a very heavy volume of UPDATE commands given in a session, $EDIT=YES is a command of convenience, sometimes of questionable merit. When this option is switched on, UPDATE will interpret every line as a command, because the computer already receives a "$" sign automatically as the first character. It causes a dilemma if you actually wants to insert a line. A command

$CREATE /TEXT/

will cause a line represented by TEXT to be inserted after the current line, and may be used for insertion when the $EDIT switch is on.

Example: A segment of editing involving $EDIT switch.

UPDATE Commands	Comments
>$TYPE 2	EDIT switch is OFF.
C Illustrative Example	
C Main Program	Two lines typed out
>$EDIT=YES	EDIT switch is now ON.
>$C WRITTEN BY T. W. SZE	Attempt to insert a line;
?>$CR/C WRITTEN BY T. W. SZE/	note error return
>$	"$" automatically given

C. $ECHO=YES; $ECHO=NO When this switch is turned on, an inserted line will be echo-printed on the terminal right after the insertion. This switch is very useful when used in conjunction with the case-shifting switch or the tab-setting switch.

Example: In the following example, a table is being constructed with data in columns 11-12, 21-22 and 31-32. Tabs are set at 11, 21 and 31. The $ECHO=YES switch will echo back entered data at the correct column positions.

UPDATE Commands			Comments
>*$TAB=11,21,31*			Set TAB.
>*$ECHO=YES*			Set ECHO switch.
>*(Tab)23(tab)55(tab)92*			Enter data.
	23	55 92	Data echoed.
>*(Tab)43(tab)12(tab)28*			
	43	12 28	
>			

D. $ERROR=YES; $ERROR=NO When this switch is turned on, an error message will be reported on the terminal when an error is committed. If the switch is turned off, only a "?" symbol is reported to indicate an error.

Example: Suppose UPDATE is editing a file that contains 5 lines. The contents of these 5 lines are 11, 22, 33, 44, and 55 respectively for each line starting at column-1. The pointer is now at line No.3. Observe the errors made in the editing session and error message received:

UPDATE Commands	Comments
>$TYPE	... Display line 3.
33	
>$CH/11/66/	... To make a change.
?>$ERROR=YES	... "?" symbol returned. Turn on
>$CH/11/66/	error message and try again.
?Sequence not in current line	... meaning can't find a match
>$CH/33/66/	... Try again.
66	... Change verified
>$TO/88/	... Move pointer.
?Reached last line of text	... Can't find /88/.
>$WHERE	... Check line number.
[5]	
>$DUNE	... Close the editing.
?Illegal command or structure	... Incorrect spelling
>$DONE	... Try again.
Catalog name=>DATA.DAT	... Name DATA.DAT given
File .... Already existing file	... Duplicate name error
Catalog name=>DATAX.DAT	... Give another name.
1 blocks written on DATAX.DAT[115103,320571]	

E. $GAG=YES; $GAG=NO After the commands TO, TRAVEL, CHANGE, SUBSTITUTE are executed, the terminal automatically prints out the new current line. While it serves as a convenience, it may become a nuisance if there is too much output. To suppress the printing, use $GAG=YES command, and the current line can only be printed out by an explicit $TYPE command. This function can be cancelled by a $GAG=NO command.

Example: To suppress unwanted verification printout:

```
>$SUBSTITUTE/ITEM1/ITEM2/
   volume
     of
 verification
 printout
>
```

```
>$GAG=YES
>$SUBSTITUTE/ITEM1/ITEM2/
>
```

F. $LINE=YES; $LINE=NO When a $LINE=YES command is given, the line number will be displayed along with the line text in all terminal displays.

Example: Observe the difference before and after the LINE switch is turned on:

UPDATE Commands	Comments
`>$AT1; TYPE 3`	Type out first 3 lines, no line numbers.
`11`	
`22`	
`33`	
`>$LINE=YES`	
`>$AT1; TYPE 3`	Type out first 3 lines with their respective line numbers.
`[1] 11`	
`[2] 22`	
`[3] 33`	
`>`	

G. $UPPER=YES, $UPPER=NO; $LOWER=YES, $LOW=NO
Many older or inexpensive terminals are built without capability of entering or outputing lower-case letters. Hence, it is often desirable to enter upper-case letters but store them as lower-cases. Since both $UPPER and $LOWER swtiches affect the cases, the aggregate effect depends on the combination of the two switches:

UPPER	LOWER	Aggregate Effect
NO	NO	Store as entered
YES	NO	Store as upper cases
NO	YES	Store as lower cases
YES	YES	store as entered

The readers are reminded that all switches ar originally at OFF or NO states. However, if there are large volume of text data containing both upper and lower cases, such as a report or a thesis, it is not practical to use this switch if one has a upper-case-only terminal. For such needs, the users are referred to the utility program RUNOFF (See Chapter 7) which contains many word-processing procedure including case-control.

(2) Format of functions switched ON temporarily by UPDATE

Frequently, it is desirable to switch certain functions ON only momentarily for the duration of one command. While the switch can always be turned on or off by commands, it will be convenient to construct a "spring-return" switch which will automatically be turned back to OFF after the command is executed. UPDATE provides this convenience by a command format in parenthesis:

```
$COMMAND (SWITCH FUNCTION) Argument
$COMMAND (FUNCTION-1) (FUNCTION-2) Argument
$COMMAND (FUNCTION-1, FUNCTION-2) Argument
```

Examples: The following examples show equivalent commands:

Equivalent		Equivalent	
$LINE=YES $TYPE 3 $LINE=NO	$TYPE(LINE)3	$ERROR=YES $TO/XYZ/ $ERROR=NO	$TO(ERROR)/XYZ/

Equivalent	
$GA=YES $ER=YES $SU/XX/YY/ $GAG=NO $ERROR=NO	$SU(GA,ER)/XX/YY/

2.14 Editing Function Value-Setting Commands

In this group of commands, the common format is:

```
$COMMAND = n
```

where "n" is an integer. The meaning of "n" is defined for each function, and they are presented as follows:

A. $FACTOR=N This command will modify the size of the memory "window". Normally, there is no need to adjust the window. Only when editing a very large file, there may be justification to adjust the window to a larger size in order to reduce the overhead file-copying when the pointer is backed outside the current window.

This was explained in a previous section in reference to Figure 2.1. When $FACTOR is given without an argument, it is an inquiry for the size of memory assigned behind the current line. A number typed out on the terminal indicates the size in number of lines.

B. $LENGTH=N, and $SIZE=N Either of the commands will set the length of each line to N characters long. If the text in a line is less than N-character long, spaces after the last character are padded with blanks. If the text in a line is more than N-character long, the extra characters are simply truncated and removed.

Complications arise when there are tabs characters in a line. Although each tab character counts only as one character in the line text, its effect is equal to multiple and variable blanks when it is translated. Therefore, if the $LENGTH or $SIZE command is used to prepare a fixed-length record file, it is desirable to let UPDATE translate tabs into blanks by $TAB command, so that a correct number of characters will be counted. The main usefulness of this command is to construct a data file in which the record size is uniform for every record. If $LENGTH command is applied without any argument, it becomes an inquiry about the length of the current line. The computer will respond with a number which is equal to the number of characters (including blanks) in the current line.

C. $SAVE=N This is a safety feature that can be very useful in long editing sessions. If the editing session in progress and the System must be re-initialized due to crash, broken communication linkage or any other emergency situation, all fruits of labor during that editing session are lost. Or, when the connect-time of the terminal expires, there will be no allowance for the user to finish or to close the editing, and he is forced to sign off immediately. The result of the current editing is also lost. If such contigency may be likely, it is prudent for a user to apply a command $SAVE=N. The following will then be accomplished:

When a command such as $SAVE=15 is issued, the output file E2 will be periodically closed, stored, and reopened to continue, for every 15 lines output into the E2. Thus, in case of a system failure, the user will in his disk a TMP file named E200xx.TMP that contains the status of last save. This would cut the loss of information to a small amount. The exact name of the TMP file is reported on the user's terminal. Should the editing goes to the completion successfully, that TMP file is deleted automatically. The disadvantage of such safety measure is that it significantly slows down the editing operation because of the extra file operations the computer is required to do every N lines.

D. $TAB=N1,n2,... When the UPDATE is first called, the tab settings are at the system default positions, namely at columns 9, 17, 25, etc (every 8 columns). To reset the values of tab setting to a different set, use the command $TAB=n1,n2,... where "n1", "n2", etc., are the new tab settings. When a tab key is subsequently entered, it will be translated into multiple blanks, the number of which depends on where the tab key is entered on the line. Since tab key often causes problem in the count of characters in a line, especially in the case of a fixed record-length file, it is useful to

use this command even though the tab settings may be the same as the system default.

The chief usefulness of this command is to prepare tables with fixed columns, or to prepare a fixed column data file.

Example: Construct a roster of names with last names starting on column-5 and initials starting on column-25:

```
>$TAB=5,25
>(T)Doe(T)JD                    (T) =Tab key
>(T)Jones(T)MS
>(T)Li  (T)JG
>(T)Kong(T)KK
>(T)Modzelewski(T)SW
>(T)Smith (T)YT
>$AT1; TYPE 6
    Doe                 JD
    Jones               MS
    Li                  JG
    Kong                KK
    Modzelewski         SW
    Smith               YT
>
```

Example: Prepare a data file that has a FORTRAN format of 2(7X,I3).

```
>$TAB=8,18
>(T)238(T) 23                (T)=TAB KEY
>(T) 12(T)856
>(T) 44(T)433
...
>$AT1; TYPE 3
       234        23
        12       856
        44       433
>
```

2.15 Miscellaneous Editing Commands

(1) Commands regarding to current line position

A. $WHERE and $LINE Either of these two commands will cause the absolute line number reported on the terminal.

B. $LENGTH This command will cause the length of the current line in number of characters reported on the terminal. Also refer to the command $LENGTH=n command. Note that $LENGTH is to inquire about the length, while $LENGTH=n is to set the length.

C. $POSITION /TEXT1/TEXT2/... This command will type out the positions (column numbers) of the first character of each of the string TEXT1,TEXT2,... in the current line.

(2) Insertion Commands While UPDATE will accept any input line without the "$" sign in column 1 to be an inserted line, there are occasions insertions may be made easier by the following commands:

A. OVERLAY /TEXT/K, or $K /TEXT/ This command will place a string of characters "TEXT" in the current line beginning at the Kth column and replacing whatever was there before.

Example: Observe the effect of a command $4/ABCD/:

Before	After
1234567890	123ABCD890

B. $PLACE/TEXT/K This command will insert the string "TEXT" in the current line starting at the Kth column. Unlike the $OVERLAY command, the displaced characters do not disappear; they are merely pushed back to the right to make room for the inserted string.

Example: Observe the effect of acommand $PLACE/ABCD/4 and compare it with that of the previous example:

Before	After
1234567890	123ABCD4567890

C. $REPLACE N When this command is given, the specified number of line in the file beggining with the current line is deleted, and the same number of lines subsequently typed on the terminal will take their palces. This command is equivalent to a compounded command of $DELETE(GAG); AT-1. The command $REPLACE is equivalent to $REPLACE 1.

Example: Observe the difference between REPLACE and DELETE commands:

$DELETE command	$REPLACE command
>$AT1; TYPE 5	*>$AT1; TYPE 5*
11	11
22	22
33	33
44	44
55	55
>$AT3; DE(GAG)	*>$AT3; RE*
>XX	*>XX*
>$AT1; TYPE 5	*>$AT1; TYPE 5*
11	11
22	22
44	XX
XX	44
55	55

(3) Length-manipulating commands

The end of a line is indicated by a carriage-return character. The number of characters between two carriage return characters, not counting the carriage return characters themselves, is the length of a line. Therefore, by adding a carriage return some place in a line, it may be broken into two lines. Conversely, if the carriage return at the end of a line is removed, that line is joined with the next. In manipulating the length in this manner, caution should be exercised regarding the blanks at the end of a line. Normally, when there are trailing blanks in a line, UPDATE simply ignores them in order to conserve storage spaces. Thus, the number of blanks at the joint should be carefully observed, otherwise the space at the "seam" will be in error. The associated commands are now discussed next.

A. JOIN command This command will remove the carriage return character at the end of the current line, thereby join it with the next line. Because all trailing blanks are deleted, any blanks required at the seam must be provided by the leading blanks of the second line in the joining process.

B. $BREAK command This command will insert a carriage return character into the current line, thereby braking it into two lines. It has two formats:

```
$BREAK N

$BREAK /TEXT/
```

Both "N" and "TEXT" indicate the end of the first line after the break. Thus, the second line after the break will begin with the old (N+1)th column as its first column, or the column immediately after the string "TEXT" as its first column.

Examples: Observe the effect of $JOIN and $BREAK. Pay attention specially to the "seam", before and after the operation.

```
>$TYPE 3
11
 22
33
>$JOIN
11 22
>$JOIN
11 2233
>
```

```
>$TYPE 2
12345 67890
>$BREAK/5/
12345
 67890
>
```

```
>$TYPE 2
12345 67890
>$BREAK/5 /
12345
67890
>
```

SELECTED ADVANCED TOPICS IN UPDATE

The materials presented in the PRIMER (pp.37-42) are for the beginning users. The materials presented in the COMMAMDS and PROCEDURES (pp.43-62) are for the average users. The combined materials should be more than adequate for most editing jobs. Occasionally, there may be special and frequent needs for very sophisticated editing and therefore a more complicated set of commands may be useful. However, unless you have special needs that require the commands in the following sections, your time may be better invested by thoroughly familiarizing yourself with the basic material and then going directly to the SUMMARY sections (page 72). It should be noted that the objectives accomplishable by the complex commands can also be accomplished by simpler commands in more steps. Or, it may require getting on and off from UPDATE several times.

Three topics will be presented: auxiliary files, conditional commands, and editing programs.

2.16 Preparation and Use of Auxiliary Files

Sometimes, it is desirable to construct an auxiliary file which contains a selected excerpts from a main file. Or, in creating a new file, certain of its lines may be contributed by an already established file. Using only commands presented so far, one can take the established file, delete all unwanted lines, and the result would be an excerpt. In this section, some special UPDATE commands are presented that will facilitate such a task.

(1) Auxiliary output file preparation

A main file is already in existence and has been called by UPDATE. It is required now to make one or more auxiliary files which contain excerpts from the main file. Three commands are provided for this purpose:

A. $ONTO command This command will open an auxiliary file in the disk into which excerpts of the main file will be transferred. The opened file will be given a filename in the ONTO command format:

$ONTO = standard file specification

where the standard file specification will contain a name and an extension.

B. $PUT N command This command will transfer N lines, beginning with the current line, from the main file to the auxiliary file opened by a previous $ONTO command. If N is omitted in the command, it is equivalent to $PUT 1. Caution: After the lines are transferred to the auxiliary file, those lines are no longer in the main file. If the editing session is allowed to end with a normal $END or $FINISH command, the new main file will be the old file minus those exerpts taken out. If you do not wish to disturb the old main file, you must not let the editing session come to a normal end. As soon as the auxiliary file preparation is completed and closed, apply CTRL-C to abort the editing job.

C. $CLOSE command — The command $ONTO opens an auxiliary file E3 as a working file; $CLOSE command closes it and stores it away in the disk.

Examples: Two auxiliary files X.DAT and Y.DAT are prepared composed of excerpts from a main file SAMPLE.DAT. Observe the sequence of editing commands and the "Before" and the "After" conditions of the files:

(BEFORE) SAMPLE.DAT	Editing Commands	(AFTER) SAMPLE.DAT	X.DAT	Y.DAT
11	*.UPDATE SAMPLE.DAT*	44	22	11
22	11	77	33	
33	*>$AT2; ONTO=X.DAT*		55	
44	*>$PU2; AT+1; PU2; CLOSE*		66	
55	*>$AT1; ONTO=Y.DAT*			
66	*>$PU1; CLOSE; END*			
77				

(BEFORE) SAMPLE.DAT	Editing Commands	(AFTER) SAMPLE.DAT	X.DAT	Y.DAT
11	*.UPDATE SAMPLE.DAT*	11	22	11
22	11	22	33	
33	*>$AT2; ONTO=X.DAT*	33	55	
44	*>$PU2; AT+1; PU2; CLOSE*	44	66	
55	*>$AT1; ONTO=Y.DAT*	55		
66	*>$PU1; CLOSE*	66		
77	*>↑C*	77		
	.			

(2) Auxiliary input file operation

Often, the input insertion to a file is preferred to be lines from an existing file if it is already available. Presumably, that file has been checked out already and it is not only convenient to copy those lines but also reduces the chances of error.

A. $FROM command — While the $ONTO command specifies a destination auxiliary file, the $FROM command specifies a source file. Its command format is similar to that of the $ONTO command:

$FROM = standard file specification

If this file resides in another user's disk area, his PPN should be a part of the file specification, such as NAME.EXT[m,n].

B. $ADVANCE command — When the $FROM command is first applied, its pointer is positioned at line 1. The $ADVANCE n command is used to position the pointer in the auxiliary file specified by the $FROM command. Although n is an unsigned integer, it is interpreted as a relative line number.

C. $GET command Once the pointer of the auxiliary file is positioned correctly in the auxiliary file, a command of $GET n will transfer n lines, starting with the current line, from the auxiliary file to the main file. After the transfer, the lines in the auxiliary file are not erased.

Sometimes, excerpts are taken from several auxiliary files. In changing from one auxiliary file to another, it is necessary to disengage the old one before engaging the new. For this reason, the command $FROM is designed to disengage automatically the old auxiliary file and engage the new file. Each time a file is engaged, the pointed will be positioned at line 1.

Examples: A file SAMPLE.DAT and two auxiliary files X.DAT and Y.DAT are all available in the disk storage. Their contents are as follows:

File Contents ---"BEFORE"

SAMPLE.DAT	X.DAT	Y.DAT
11	AA	XX
22	BB	YY
33	CC	ZZ
44	DD	UU
55	EE	VV
66		
77		

Another file Z.DAT is now prepared by inserting certain lines from X.DAT and Y.DAT into SAMPLE.DAT. This is shown below:

Editing Commands	File Contents --- "AFTER" SAMPLE.DAT	X.DAT	Y.DAT	Z.DAT
.UPDATE SAMPLE.DAT	11	AA	XX	11
11	22	BB	YY	22
>$AT2; FROM=X.DAT	33	CC	ZZ	BB
>$ADVANCE 1; GET 2	44	DD	UU	CC
>$AT/55/; FROM=Y.DAT	55	EE	VV	33
>$AD2; GET 2: DONE	66			44
	77			55
CATALOG NAME=>*Z.DAT*				ZZ
1 BLOCK WRITTEN ON Z.DAT				UU
				66
				77
EXIT				
.				

2.17 Conditional Editing Commands

The UPDATE is enhanced in capabiltiy by being able to make "decision" on which one of two alternate groups of editing commands are to be executed.

The basic structure of decision-making is as follows: First, a question is asked to which a true-false answer is stored. This is accomplished by issuing a $IF command. If the answer is affirmative, issuing a $THEN command will execute a group of "execute-if-true" eiditng commands. (If the answer is

negative, issuing a $THEN command will receive no response from them.) If the answer is negative, issuing a $ELSE command will execute a group of "execute-if-false" editing commands. (Similarly, if the answer is affirmative, issuing a $ELSE command will receive no response from them.) Such a structure is simialr to the conditional structure in many language processors, and is graphically illustrated in a flow chart as shown in Figure 2.2.

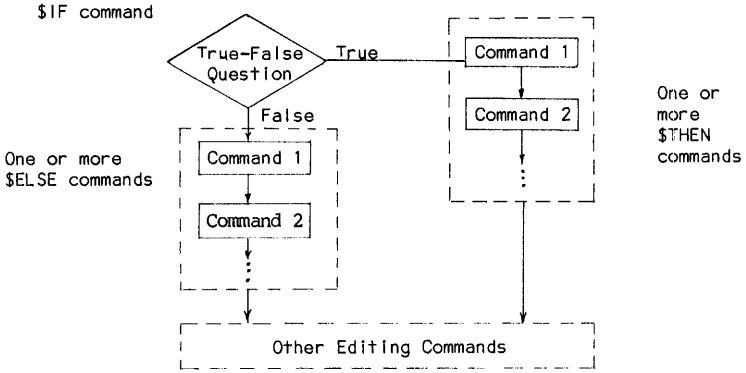


Figure 2.2 Flow Chart of Conditional Editing Commands

(1) Single conditional commands

A. $IF command As illustrated in Figure 2.2, the $IF command asks a true-false question, and its answer is stored away, setting the stage for subsequent actions of $THEN and $ELSE commands. Since the UPDATE has immediate information only on the current line, the question asked must pertain either to the current line number or to its content. Therefore, the formats of the $IF command are limited to the following:

$IF format	Question Asked
$IF /TEXT/	Is there a string "TEXT" in the current line?
$IF /TEXT/K	Is there a string "TEXT" in the current line that begins at the Kth column?
$IF $TEXT$	Ignoring blanks, tabs, and difference between upper and lower cases, is there a string "TEXT" in the current line?
$IF n	Is the current line number equal to n?

Notice that the formats of the first three are very similar to those of $TO commands.

B. $THEN and $ELSE commands The $THEN and the $ELSE commands will specify and execute the alternate sets of commands depending on the answers to a previously issued $IF command. The command format is as follows:

$THEN /command 1; command 2; .../

$ELSE /command 1; command 2; .../

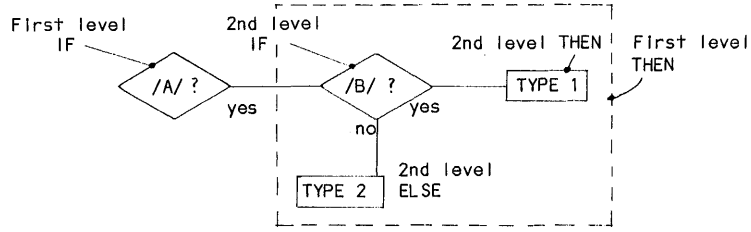
The commands between the delimiters "/" follow the rules of compounded command structure, as discussed in section 2.10.

Example: $IF/FORMAT/7; THEN/WHERE; TYPE 2/;ELSE/DELETE 2/

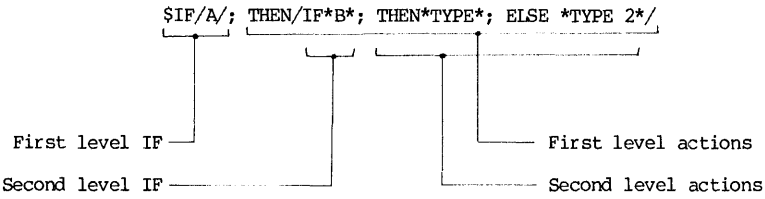
Function: Examine the current line. Does it have a string of characters "FORMAT" beginning at the 7th column? If yes, print out the line number and type two lines. If no, delete 2 lines.

(2) Nested conditional commands

Each of $THEN and $ELSE command contains a set of embedded commands in the compounded form. If the embedded commands contain another IF command, we now have a nested structure. The following flow chart shows a typical example of nested command structure:



The function of this flow chart is as follows: First examine the current line to see if there is a character "A". If no, do nothing. If yes, then examine if there is also a character "B" in the current line. If yes, type one line; if no, type 2 lines. These functions may be accomplished by the following nested command:

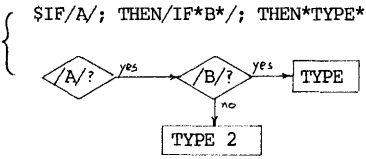


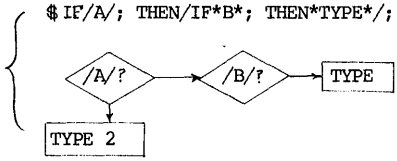
In using a nested conditional structure, one should be cautioned about the following:

A. The main advantage of the nested conditional structure is to compress many editing commands into a single compounded one, so that its execution will be more efficient. The UPDATE allows a maximum of ten nesting levels. The main drawback is that constructing a nested structure is a very error-prone process. Furthermore, more levels it goes into, less man-machine interaction is available to the user. Therefore, even though the UPDATE is machine-effective for high-level nesting, it is a poor practice for a user to go much beyond the second level. Otherwise, an editing session will be very likely degenerated into a debugging session for editing commands. An exception to this advice is when one has some nested commands that will be used repeatedly by the user or others. In such a case, it may be justifiable to spend a lot of time to debug it and store it for later repeated use.

B. In addition to the logic involved, the most likely source of error in a nest construction is the choice of delimiters. Normally, any special symbol pair may be used as delimiters (or as "quotation marks"). Since a nested structure is basically a compounded structure, the semicolons ";" must be reserved to separate the commands. Moreover, there should be no ambiguity between the command delimiters of IF, THEN, ELSE at different levels. Therefore, it is advisable to assign an unique delimiter symbol for each level. See the following illustrative examples:

Example: Consider the following nested commands with their respecitve interpretation of functions by means of flow chart:





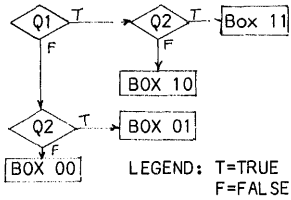
IF/A/; THEN/IF/B/; THEN/TYPE//; ELSE /TYPE 2/
Incorrect use of delimiters!

C. Several nested commands may be compounded together to form a compounded nested command. In doing so, one must be careful about the correct placement of the THEN, ELSE commands. Each time when a first-level IF command is executed, its true-false answer replaces that obtained in a previous IF command. The same goes for the subsequent level IF commands. Thus in the above example, the actions of both statements may be combined by this statement:

$IF/A/; THEN/IF*B*; TH*TY*;EL*TY2*/;TH/IF*B*;TH*TY*/;EL*TY2*

(3) Conditional with logic connectives

Consider the following fully-developed two-level nested structure:



Logic Connective Between Q1 & Q2	Accomplished by Placing the same actions in Box 00	01	10	11
AND				X
OR		X	X	X
NAND	X	X	X	
NOR	X			
XOR		X	X	

By placing identical editing actions in the appropriate boxes as shown in the accompanying table, a logical connective between the answers to Q1 and Q2 may be accomplished. For example, if one wants to type the line if either character "A" or character "B" (or both) is present, he should place the TYPE command in boxes 01, 10 and 11. The result is the following command:

$IF/A/; THEN/IF*B*; THEN *TYPE*; ELSE *TYPE*/; ELSE/IF*B*; THEN *TYPE*/

Actually, one can see that there is an INCLUSIVE OR, or logical union relation existed in this case. The UPDATE processor has simplified the matter by providing five commands specifying logic connectives: AND, OR, NAND, NOR and

XOR. They are respectively for logic intersection, logic union, negation of AND, negation of OR, and EXCLUSIVE OR. Thus, the above example can now be written as:

```
$IF/A/; OR/B/; THEN /TYPE/
```

There is one important caution in using the logic connective commands. Unlike the two-level nested commands, the second-level question does not establish an independent answer (True-False) but modifies the first one. Therefore, if the first-level question alone is to initiate some THEN or ELSE action, it better be done before the answer is changed by the logical connective commands. Observe the difference between the following two editing commands:

```
$IF/A/; ELSE /TYPE/; OR /B/; THEN /DELETE/

$IF/A/; OR /B/; ELSE/TYPE/; THEN /DELETE/
```

The difference would be the execution of ELSE/TYPE/ segment.

2.18 Editing Programs

In the UPDATE processor, the compounded command structure enables a series of command executions in one pass. The TRAVEL, GO, and STOP commands result in the looping capability. The conditional command group IF, THEN, ELSE, and logic connectives yield the decision-making capability. Combining all of these, one has the makings of a complete stored editing program. However, it is not always desirable to construct editing programs for one-shot usage as they are very wasteful of user resources. Moreover, accuracy of editing requires a high degree of user-machine interaction which a complete editing program will deprive. Therefore, construction of editing programs should be limited to applications of wide and frequent usages.

Two such program are given as illustrative examples:

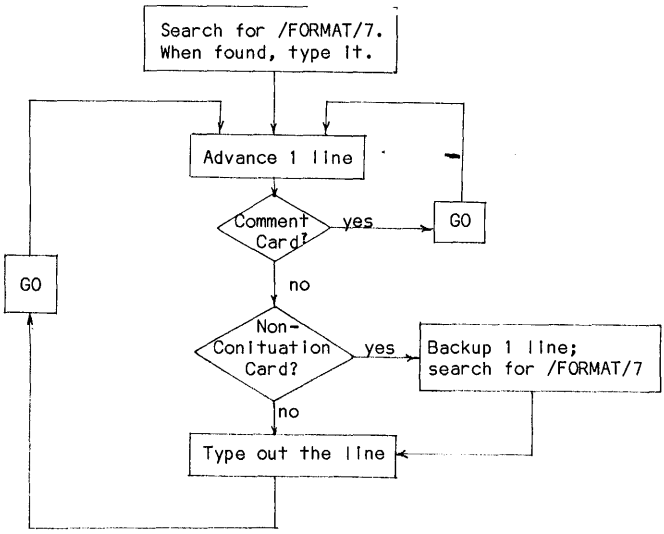
Example: Given a FORTRAN program, design an editing program that will print out all FORMAT statements. Assume all FORMAT statements have the keyword "FORMAT" beginning at column 7, but some of the FORMAT statement may have continuation cards.

The logic of the program may be described by means of a flow chart shown on the next page.

In the compounded structure form, the resulting program (one line) is as follows:

```
$GAG=YES;AT/FORMAT/7;TYPE;TR+1; IF/C/1;OR/*/1; THEN/GO/; IF/ /6;
    THEN/AT-1; AT'FORMAT'7/; TYPE; GO
```

After this command is completed, all FORMAT statements will have been typed out on the user's terminal. An error report sign "?" will also be typed out, because when the search reaches the end of file, the $TR+1 command will still attempt to advance 1 line. If the above program has PUT commands instead of TYPE commands (with ONTO command issued previously), this program would have prepared an auxiliary file that contains all FORMAT statements in the FORTRAN program.



Example: The equivalence between the 026 and the 029 key punch code is shown below:

026 Punch	029 Punch
#	=
&	+
@	'
%	(
<	)

Other characters have the same punch codes.

```
$TR+1;IF/ /;TH/AL. .;AT-1/;OR/&/;TH/AL.&.+.;AT-1/OR/@/;TH/AL.@.'.;
    AT-1.;OR/%/;TH/AL.%.(.;AT-1/;OR/</;TH/AL.<.).;AT-1/;GO
```

Since a single-line compounded command is limited to a maximum of 150 characters, two-letter abbreviations are used for all UPDATE keywords in the above command.

A SUMMARY OF FILE MANAGEMENT BY UPDATE

2.19 File management Tasks

(1) To create a new file from a terminal

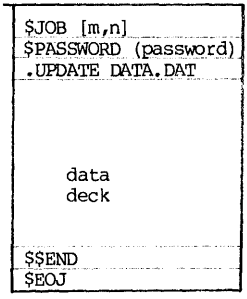
When UPDATE receives the input file name, the disk storage directory is searched. when the file is found, the file is loaded into the input working file.

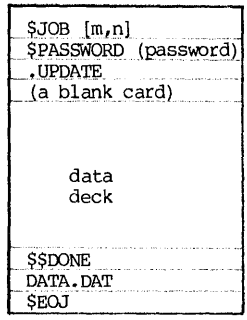
However, if the file name supplied by the user is null (represented by a carriage return and nothing else), or if the file does not exist for the name given, the input working file is entirely blank. Thus, the only information that may go to the output working file would be from the terminal or from other stored files by insertion mode. In this way, an entirely new file may be created from the user's terminal and stored in the disk.

(2) To create a new file by batch

The effectiveness of UPDATE to do editing job is mainly because its man-machine interaction. Therefore, UPDATE normally is not suitable for BATCH jobs. However, if the source materials are in punched card form, a file may be created from these cards by UPDATE submitted in BATCH.

Suppose we wish to store a deck of data cards in disk and will name the file as DATA.DAT. First, a batch deck of cards is prepared that contains the following. Either will do:





In the above deck setup, the single-$ cards are BATCH commands, and those double-$ cards are UPDATE commands read by BATCH. For more details on Multiprogram Batch, see Chapter 9.

After the cards are prepared, read the cards in at a RJE card reader. The job will be executed by the computer, and the file DATA.DAT is thus created from the cards. For card input used in this way, the same precaution should be exercised that there should be no "$" character in the first column in the data card deck.

(3) To copy a file

A file may be duplicated and stored in the user's disk area by using the UPDATE in the folowing way:

```
.UPDATE NAME.EXT[115103,320571]
(UPDATE prints out the first line of NAME.EXT)
>$DONE
CATALOG NAME=> NEW.EXT
```

This is equivalnet to a monitor command of:

```
.COPY NEW.EXT = NAME.EXT[115103,320571]
```

(4) To merge several files into one

For example, if three files D1.FOR, D2.FOR and D3.FOR are to be merged into one DX.FOR, it can be accomplished in the follwoing way:

```
.UPDATE D1.FOR
(UPDATE prints out the first line of D1.FOR)
>$AT $
>$INPUT= D2.FOR
>$INPUT= D3.FOR
>$DONE
CATALOG NAME=>DX.FOR
```

This is equivalent to applying the monitor command:

```
.COPY DX.FOR = D1.FOR,D2.FOR,D3.FOR
```

(5) To prepare an auxiliary file from a source file

The following is an example where an auxiliary file FORMAT.FOR is prepared by extracting all FORMAT statements (some of which may be multiple-line statements) from the FORTRAN file SAMPLE.FOR. Assume that all keyword FORMAT of the FORMAT statements starts at the 7th column.

```
.UDPATE SAMPLE.FOR
(UPDATE prints out the first line of SAMPLE.FOR)
>$BEFORE
>Δ                                                    (Δ=blank)
>$ONTO=FORMAT.FOR
>$GAG=YES;AT/FORMAT/7;PU;TR+1;IF/C/1;OR/*/1;TH/GO/;IF/ /6      one
    TH/AT-1;at'FORMAT'7/;PU;GO                                 line
?>$CLOSE
>↑C                    ?=error indication when reaching the end
                         of file and still wanting to "GO"
```

The logic of the long command line in this example was discussed in Section 2.18.

2.20 Examples of File Editing

Two examples of editing a complete file will be given using the UPDATE editor. The first one consists of entirely text eidting, while the second one is a stored program in FORTRAN. The following points will be helpful:

(1) A careful proof-reading of the old text is essential. It is also desirable to do the proof-reading "off-line" to conserve valuable terminal time.

(2) To increase the speed and efficiency of editing (and therefore to reduce time and cost), all corrections should be marked on the listing, together with their line numbers if appropriate.

(3) Moving from one record to another, the normal operation of the UPDATE editior is to go forward. In fact, backing up the pointer to some previous line may sometimes be costly because it will involve file re-writing and re-reading. Therefore, backing up is generally an inefficient process and should be used sparingly in view of processor efficiency. On the other hand, since deletions and insertions of lines during editing will change the line numbers of all lines of text beyond the pointer, it will be progressively difficult to locate the desired line by absolute line numbers. For this reason, in editing the text by its absolute line numbers, it is sometimes desirable that the editing be done in the reverse direction, starting from the end of the file and working backwards toward the front. In this manner, the deletion and insertion of lines will not affect the line numbers of the portions of the file not yet edited. Here we are trading off machine and processor efficiency for user convenience. This process is desirable only if the user has made preparations as outlined in (1) and (2). To improve processor efficiency, he can also readjust and enlarge the window by the $FACTOR command.

(4) Only the first two letters of any UPDATE command word need be given. Incorrect spelling of command is tolerated as long as the first two letters are spelled correctly.

Example 1: To edit the text taken from the School of Engineering Bulletin, University of Pittsburgh. The draft of text on disk file TEXT.EDT along with the revisions on the draft appears as follows:

```
THE MICHAREL L. BENEDUM HALL OF ENGINEERING

    STDUENTS ENROLED IN THE SCHOOOL OF ENGINEERING, UNIVER TY OF
PITTSBURGH, RECEIVE THEIR EDUCATION IN ONE OF THE COUNTRY'S MOST
MODERNAND BEST EQUIPPED ENGINEERING BUILDINGS, THE MICHARL L. BENNEDUM
HALL OF ENGINEERING. THE BUILDING COMPLEXX IS NAMED IN HONOR OF
MICHAEL L. ENEDUM, A PIONEER IN THE OIL INDUSTRY AND CO-FOUNDER OF
MICHAEL L. ENEDUM, A PIONEER IN THE OIL INDUSTRY AND CO-FOUNDER OF
THE BENEDUM TREE OIL COMPNAY. A GRANT FROM THE CLAUDE WORTHINGTON
FOUNDATION ENABLED THE UNIVERSITY TO PURCHASE ON WHICH THE ENGINEERS
COMPLEX IS BUILT.
```

The following is a printout of the editing session:

```
.UPDATE TEXT.EDIT
THE MICHAREL L. BENEDUM HALL OF ENGINEERING
>$CH/T/          T/R//
          THE MICHAEL L. BENEDUM HALL OF ENGINEERING
>$AT+2;CH/L/LL/000/00/R TY/RSITY/
     STUDENTS ENROLLED IN THE SCHOOL OF ENGINEERING, UNIVERSITY OF
>$AT+2;CH/AND/ AND/ARL/AEL/NN/N/
MODERN AND BEST EQUIPPED ENGINEERING BUILDINGS, THE MICHAEL L. BENEDUM
>$AT+1;CH/X//
HALL OF ENGINEERING.  THE BUILDING COMPLEX IS NAMED IN HONOR OF
>$AT+1;CH/EN/BEN/
MICHAEL L. BENEDUM, A PIONEER IN THE OIL INDUSTRY AND CO-FOUNDER OF
>$AT+1; DELETE
THE BENEDUM TREES OIL COMPANY.  A GRANT FROM THE CLAUDE WORTHINGTON
>$AT+1;CH/CHASE/CHASE THE LAND/ENGINEERS//
FOUNDATION ENABLED THE UNIVERSITY TO PURCHASE THE LAND ON WHICH THE
>$AT+1; PLACE/ENGINEERING /1
ENGINEERING COMPLEX IS BUILT.
>$END
>
1 Blocks written on TEXT.EDT[33,33]

EXIT
```

The edited file is shown below:

THE MICHAEL L. BENEDUM HALL OF ENGINEERING

STUDENTS ENROLLED IN THE SCHOOL OF ENGINEERING, UNIVERSITY OF PITTSBURGH, RECEIVE THEIR EDUCATION IN ONE OF THE COUNTRY'S MOST MODERN AND BEST EQUIPPED ENGINEERING BUILDING, THE MICHAEL L. BENEDUM HALL OF ENGINEERING. THE BULDING COMPLEX IS NAMED IN HONOR OF MICHAEL L. BENEDUM, A PIONEER IN THE OIL INDUSTRY AND CO-FOUNDER OF THE BENEDUM TREES OIL COMPANY. A GRANT FROM THE CLAUDE WORTHINGTON FOUNDATION ENABLED THE UNIVERSITY TO PURCHASE THE LAND ON WHICH THE ENGINEERING COMPLEX IS BUILT.

Example: To edit a stored FORTRAN program. It is suggested that the readers follow the running comments marked on the printout.

```
.UPDATE SAMPLE.FOR
[CREATE NEW FILE]
>C     SAMPLE PROBLEM FOR THE TIME-SHARING NOTES
>       READ(5,10)A,B,C,D,X1
>   10 FORMAT(F20.7)
>    1 X2=X1-(A*X1**3+B*X1**2+C*X1+D)/(3.*A*X1**2+2.*X1+D)
>      WRITE(6,10)X2
>       IF(ABS((X1-X2)/X2-.001)3,3,2
>    2 X1=X2
>      GO TO 10
>    3 WRITE(6,11)X2
>   11 FORMAT(/' THE REAL ROOT = ',F20.7)
>      STOP
>      END
>$wh
12
```

```
>$AT1; TYPE 12
C     SAMPLE PROBLEM FOR THE TIME-SHARING NOTES
      READ(5,10)A,B,C,D,X1
   10 FORMAT(F20.7)
    1 X2=X1-(A*X1**3+B*X1**2+C*X1+D)(3,*A*X1**2+2.*X1+D)
      WRITE(6,10)X2
      IF(ABS((X1-x2)/X2-.001)3,3,2
    2 X1=SX
      GO TO 10
    3 WRITE(6,11)X2
   11 FORMAT(/' THE REAL ROOT = ',F20.7)
      STOP
      END
>$TO4
    1 X2=X1-(A*X1**3+B*X1**2+C*X1+D)(3,*A*X1**2+2.*X1+D)
>$CHANGE $D)(3,$D)/3.$2.$2.*B$D$C$
    1 X2=X1-(A*X1**3+B*X1**2+C*X1+D)/(3.*A*X1**2+2.*B*X1+C)
>$AT+1; CH/W/ W/
      WRITE(6,10)X2
>$AT+1; CH/X2-/X2)-/
      IF(ABS((X1-X2)/X2-.001)3,3,2
>$AT+1; CH/SX/X2/
    2 X1=X2
>$AT+1; CH/0//
      GO TO 1
>$END
1 Blocks written on SAMPLE.FOR[33,33]

EXIT

.TYPE SAMPLE.FOR
C     SAMPLE PROBLEM FOR THE TIME-SHARING NOTES
      READ(5,10)A,B,C,D,X1
   10 FORMAT(F20.7)
    1 X2=X1-(A*X1**3+B*X1**2+C*X1+D)/(3.*A*X1**2+2.*B*X1+C)
      WRITE(6,10)X2
      IF(ABS((X1-X2)/X2-.001)3,3,2
    2 X1=X2
      GO TO 1
    3 WRITE(6,11)X2
   11 FORMAT(/' THE REAL ROOT = ',F20.7)
      STOP
      END
```

EXERCISES

1. (a) Enter the following FORTRAN program in your disk by using UPDATE and name the file as PROB1.FOR:

```
C   PROBLEM NO. 1
      DIMENSION K(10)
      DO 5 I=1,10
5     K(I)=I**2
      WRITE(6,10)((I,K(I)),I=1,10)
10    FORMAT(2I7)
      STOP
      END
```

Purposely make some errors in your typing. For example, omit some commas and misspell a few words.

(b) When you are back at the monitor level, execute the incorrect program by a command:

.EXECUTE PROB1.FOR

and observe the proceedings.

(c) Make appropriate corrections, and execute again. Repeat until you get the program letter perfect.

2. What would each of the following UPDATE fragments do?

(a)
```
$AT 1
$SUB/XX/YY/
```

(b)
```
$AT 1
$TR/XX/
$CH/XX/YY/
$AT -1
$GO
```

(c)
```
$AT 1
$TR/XX/
$CH/XX/YY/
$GO
```

(d)
```
$SUB/READ(5,/READ(1,/
```

(e)
```
$TR+1;IF/READ/7;THEN/TYPE(LI)/;GO
```

(f)
```
$ONTO= READ.FOR
$TR+1; IF/READ/7; THEN/PUT/; GO
$CLOSE
```

3. Three different compounded MOVE commands are given:

```
$MOVE; HERE

$MOVE; AT-1; HERE

$MOVE; AT$; HERE
```

For each of these three commands, answer the following questions:

(a) Where is the line moved to?

(b) Where will be the pointer after the move?

4. Verify your answers to problem 3 by actually setting up a file, observing the BEFORE and AFTER of each of the above three commands.

5. Enter Lincoln's Gettisburg Address as a file and name it as ABE.DOC. Correct any error in the file.

6. For each line of ABE.DOC prepared in problem 5, edit the text so that the following results are obtained:

   a. Set the left margin at column 1; the right margin at column 45.

   b. The first line of a new paragraph is indented 5 spaces.

   c. Right justify by adding spaces between words.

   d. Space all punctuations so that there is one space after each comma or semicolon, and 3 spaces after each period.

7. After copying the file SYS:NEWS (see Exercise(3), Chapter 1) into your own disk area, use UPDATE and with one compounded instruction, search and type out all first lines of news items that were dated in 1980.

8. The instructor will furnish for this exercise a long FORTRAN program that contains many FORMAT, READ, WRITE and CALL statements. Prepare four auxiliary files that will contain the following information:

   (a) File FORMAT.FOR: a record of all FORMAT statements
   (b) File READ.FOR: a record of all READ statements
   (c) File WRITE.FOR: a record of all WRITE statements
   (d) File SUBR.FOR: a record of all subroutine CALL statements

   For a simple case, make the following assumptions:

   (1) All characters are upper cases.
   (2) All statement keywords begin on column 7.
   (3) No continuation statement.

9. For a more challeging case of problem 8, make the following assumptions and then prepare the required auxiliary files:

   (1) Mixed upper and lower cases in the FORTRAN program file.
   (2) A statement may begin anywhere between column-7 and column-72.
   (3) Some of the READ, WRITE or CALL statements may be imbedded in an IF statement, e.g., IF(I.EQ.1)READ(5,56)X
   (4) There may be continuation statements.

   You may modify this problem and generate a problem a varying degree of difficulty by selecting one or more of these assumptions.

10. The source program in FORTRAN-10 on DEC System-10 allows a special use of the tab key (or the CTRL-I character) to skip all or part of the label field. The purpose is to use a tab-character (1 character) to replace multiple spaces (multiple characters) to save storage space. Rules of interpreting a FORTRAN-10 statement using a tab in the initial field are as follows:

    (1) If the tab is immediately followed by one of the digits 1 through 9, that line is a continuation line of the previous one. The non-zero numeric character following the tab is considered in column-6.

    (2) Otherwise, the line is an initial line of a FORTRAN statement, and the character following a tab is considered to be in column-7.

    For example, both of the following versions of a source program are acceptable by DEC System-10:

Version 1	Version 2
C SAMPLE PROBLEM	C SAMPLE PROBLEM
bbbbbbDO 10 I=1,20	(T)DO 10 I=1,20
bbbbbbK=I**3	(T)K=I**3
bbb10 TYPE 20, I,	10(T)TYPE 20, I,
bbbbb1K	(T)1K
bbb20 FORMAT(2I12)	20(T)FORMAT(2I12)
bbbbbbEND	(T)END
b=blank space	(T)=tab

    For a FORTRAN-10 program entered by using the tab-key storage-saving technique, repeat problems 8 and 9.

11. For each of the following functions, write a single-line compounded UPDATE command to accomplish it:

    (1) To type out only those lines in a FORTRAN program that have lengths longer than 72 columns. The printout should contain absolute line numbers, line lengths, and the line itself. Do not print out all lines.

    (2) To insert the word EXERCISE in columns 73-80 of every line in a FORTRAN program.

    (3) to print out all subroutine call statements in a FORTRAN program.

    (4) To print out all FORMAT statements.

    (5) To print out all COMMENT statements.

REFERENCES

1. PTSS TEXT EDITOR, Class Notes of a Freshman Course "Engineering Analysis 2", T. W. Sze, University of Pittsburgh, Pittsburgh, Pennsylvania; 1969.

2. A PRIMER FOR PITT TIME-SHARING SYSTEM (PTSS), Chapter 5, Text Editor, T. W. Sze, University of Pittsburgh, Pittshurgh, Pennsylvania; 1970.

3. INTRODUCTION TO A TIME-SHARING SYSTEM, Chapter 6, Text Editor, T. W. Sze, University of Pittsburgh, Pittsburgh, Pennsylvania; 1972.

4. UPDATE Reference Card, Computer Center, University of Pittsburgh, Pittsburgh, Pennsylvania; June, 1979.

5. UPDATE/X - UNIVERSITY OF PITTSBURGH DATA AND TEXT EDITOR, Computer Center, University of Pittsburgh, Pittsburgh, Pennsylvania; 1976.

6. INTRODUCTION TO DEC SYSTEM-10: TIME-SHARING AND BATCH, T. W. Sze, Chapter 6, Text Editor, University of Pittsburgh, Pittsburgh, Pennsylvania; First Edition, 1974; Second Edition, 1977.

7. UPDATE, Gerald W. Bradley, Computer Center, University of Pittsburgh, Pittsburgh, Pennsylvania; 1979.

CHAPTER 3

FORTRAN-10

FORTRAN is the most widely studied and used programming language in the United States. Therefore, this chapter is prepared with the assumption that the readers already have some background knowledge of the language. For those who are not familiar with the language, please consult any one of many FORTRAN manuals available. Two typical ones are:

PROGRAMMING WITH FORTRAN, Byron S. Gottfried, Quantum Publishers, New York, 1972

PROBLEM SOLVING AND STRUCTURED PROGRAMMING IN FORTRAN, F. L. Friedman & E. B. Koffman, Addison-Wesley Publishing, Reading, Massachusetts; 1977

INTRODUCTION

There are not just a few versions of FORTRAN; there are dozens. Even on the DEC System-10 alone, there are several versions available. In attempting to unify all versions of FORTRAN developed in the computer industry, the American National Standards Institute (ANSI) in 1966 set up a standard for FORTRAN, now known as the "1966 ANSI Standard."* However, what has happened since is that the computer industry has used the Standard only as a minimum standard, and every company has extended far beyond that minimum for their own versions of the FORTRAN language. Unfortunately, while the ANSI standard part is uniform, the enhanced parts among different versions are not. Programs written in one enhanced version may require some modifications if run on a machine using a different compiler. The version of FORTRAN covered in this chapter, called FORTRAN-10 by the Digital Equipment Corporation, is a powerful superset of the ANSI standard version. A summary of FORTRAN-10 will be included in this chapter. However, readers are encouraged to seek more details from References 3 and 4.

*USA Standard FORTRAN (x3.9-1966), American National Standards Institute, 1966

RUNNING A FORTRAN PROGRAM ON DEC SYSTEM-10

After a FORTRAN program is written and thoroughly checked for its logic, running the program will require two major steps:

(1) To enter and store the FORTRAN program as a disk file.

(2) To compile, load, and execute the stored program.

The following discussion will be devoted to these two steps.

3.1 To Enter and Store a FORTRAN Program

In the DEC System-10 , the source program in FORTRAN, ALGOL, COBOL or MACRO should first be stored as a disk file because the most common way of execution is for the System to search in the disk for a specified program.

There are a number of utility programs available by which a user can enter and store his FORTRAN program. By far the best way is to use the UPDATE editor, which enables a user full editing facilities while entering a program. The details of UPDATE are given in Chapter 2. In this chapter, only the procedure relating to the entering of a FORTRAN program will be demonstrated.

As an illustration, let us consider two programs: one containing just the main program, and the other a main program plus a subroutine. The program listings are as follows:

Program 1

```
C SAMPLE PROGRAM 1
      DO 10 I=1,10
      K=I**3
   10 TYPE 20, I,K
   20 FORMAT(2I10)
```

Program 2

```
C  SAMPLE PROGRAM 2, WITH SUBROUTINE
      ACCEPT 10, M,N
   10 FORMAT(I3)
      CALL CUBE(M,N)
      END

C  SUBROUTINE FOR SAMPLE PROGRAM 2
      SUBROUTINE CUBE(M,N)
      DO 10  I=M,N
      K=I**3
   10 TYPE 20, I,K
   20 FORMAT(2I10)
      RETURN
      END
```

(1) To enter program by the UPDATE editor

The UPDATE editor was originally developed for the Pitt Time-Sharing system (PTSS) using an IBM/360 Model 50 computer, and has since been adapted for use on the DEC System-10. It enables a user not only to enter and store a program, but also to correct errors and to edit. The following shows a typical session with the UPDATE editor. The user's typings are shown in italics.

At a User's Terminal	Comments
.R UPDATE	Call for the editor
INPUT=> ↵	↵ = RETURN key of terminal
>C SAMPLE PROGRAM 1	
> DO 10 I=1,20	
*> K=I**3*	> = prompt from the computer
> 10 TYPE 20, I,K	
> 20 FORMAT(2I12)	Enter FORTRAN program
> END	
>$DONE	
CATALOG NAME=>*PRG1.FOR*	
6 BLOCKS WRITTEN ON PRG1.FOR[115103,320571]	

In a similar way, Program 2 may be entered and stored. Let us assume that the main program of Program 2 is stored and named as PRG2.FOR and its subroutine as CUBE.FOR.

If a program requires several subroutines, each subroutine may be entered and stored separately as a single file bearing a different name, or they may be combined into one file with one filename. At this point of the illustration, three files have been stored and they are PRG1.FOR, PRG2.FOR and CUBE.FOR. A listing of the programs can be made by using a monitor command:

.TYPE PRG1.FOR, PRG2.FOR, CUBE.FOR

The listings produced may be used as records or for proof-reading.

If the listing shows that the programs have been correctly entered, the programs are ready for compiling, loading and execution.

(2) To enter program via punch cards

The way to enter and store a program deck is to submit it by a batch job. Details of batch jobs are given in Chapter 9. Repeating the example above, the control file deck is first assembled as follows:

```
$JOB [115103,320571]
$PASSWORD DEBBIE
$DECK PRG1.FOR
  Program 1 deck
$DECK PRG2.FOR
  Main program deck
  Program 2
$DECK CUBE.FOR
  Subroutine deck
$EOD
$EOJ
```

```
$JOB [115103,320571]
$PASSWORD STEVE
.UPDATE PRG1.FOR
   Program 1 deck
$$END
.UPDATE PRG2.FOR
   Main program deck
   Program 2
$$END
.UPDATE CUBE.FOR
   Subroutine deck
$$END
$EOJ
```

There is often a need to enter and store a FORTRAN program via a punch card deck. For example, a card deck may have already been prepared. Perhaps the terminals are not available. Although there are more terminals than key punches, the latter are often less in demand and hence more available. After

the deck is assembled as shown in either makeup, the assembled deck is read by a system card reader, and the batch job is submitted. After the job is executed, there should be files PRG1.FOR, PRG2.FOR and CUBE.FOR in this user's disk area.

3.2 To Edit a Stored FORTRAN Program

If any typographical error, missing lines, or duplications are found in the listings of stored programs, the UPDATE editor may be used to make corrections. Suppose the PRG2.FOR listing is produced as follows and errors were found and marked as shown below:

```
C  SAMPEL PROGRAM 2, WITH SUBROUTINE
      ACCEPT 10, M,N
10    FORMAT(I3)
      CALL CUBE(M,N)
      END
```

Move over one space

END ... Missing

To make corrections, the UPDATE editor may be used either on a terminal or in a batch job:

(1) Using UPDATE at a terminal

The following represents a terminal session of error correction:

```
.UPDATE PRG2.FOR
C SAMPEL PROGRAM 2, WITH SUBROUTINE
>$CHANGE/PEL/PLE/
C SAMPLE PROGRAM 2, WITH SUBROUTINE
>$AT+2; CHANGE/FOR/ FOR/
   10 FORMAT(I3)
>$AT+1; CHANGE/M,N/M,N)/
      CALL CUBE(M,N)
>      END
>$END
```

After the editing session, the listing should again be typed out for a final verification.

(2) Using UPDATE in a batch job

Assemble a batch job deck as follows. Notice that the order of the cards and their contents are identical to those input lines in the terminal session, with the exception that an UPDATE $-command should be punched as a $$-card.

```
$JOB [115103,320571]
$PASSWORD DEBBIE
.UPDATE PRG2.FOR -
$$CHANGE/PEL/PLE/
$$AT+2; CHANGE/FOR/ FOR/
$$AT+1; CHANGE/M,N/M,N)/
      END
$$END
$EOJ
```

There is, of course, a third way: Noting the errors on PRG2.FOR, repunch the incorrect cards. Insert any missing card. Resubmit the corrected deck as a new batch job. In the batch deck, include a command first to delete the old PRG2.FOR before storing the new PRG2.

3.3 To Compile, Load and Execute a Stored FORTRAN Program

The sequence of executing a FORTRAN-10 program is as follows:

(1) To compile the specified source programs and store the binary object or relocatable files (with extensions of REL) in the disk.

(2) To load the REL files of the main program and all subprograms or subfunction programs called by the program into the core memory.

(3) To begin the execution of the loaded object program from an address determined by the compiler and the loader.

All these steps can be accomplished in sequence by a single monitor command:

.EXECUTE list

where "list" is a list of all FORTRAN programs (or their REL files if available) including any other subprogram files needed for execution in one problem. If a program belongs to another user but is accessible, the PPN of the owner should be specified along with the filename. If the file is on tape which is already mounted, then the device name should also be specified. Thus, to execute Programs 1 and 2 respectively, issue the following commands:

```
.EXECUTE PRG1.FOR
.EXECUTE PRG2.FOR, CUBE.FOR
```

When an EXECUTE command is issued, the computer will go through a sequence of compiling, loading and execution. The sequence of operations to carry out the command EXECUTE PRG2.FOR, CUBE.FOR is represented by the flow chart shown in Figure 3.1. Note particularly the processing logic by which any unnecessary compiling is avoided.

When a source FORTRAN program is compiled for the first time, a REL file is created and stored. In the user's file directory, pertinent information are also stored, such as the creation time accurate to the minute. When the program is executed again and if the program has not been modified in any way, the REL file is still valid, and compiling again would be superfluous. On the other hand, if the program has been modified since the last compiling, then the existing REL file is not valid, and compiling again during the next execution is necessary. The processing logic does it by comparing the creation time between the source porgram and its REL file. If the creation time of the source is earlier, then the REL file is still valid. If the creation time of the source is later, then the REL is not valid, and compiling should be done again. After a new REL file is created by the re-compiling, its creation time is updated also. This logic is handled by the System and the user is spared the decision. Execution of Program 1 and Program 2 are given below as illustration:

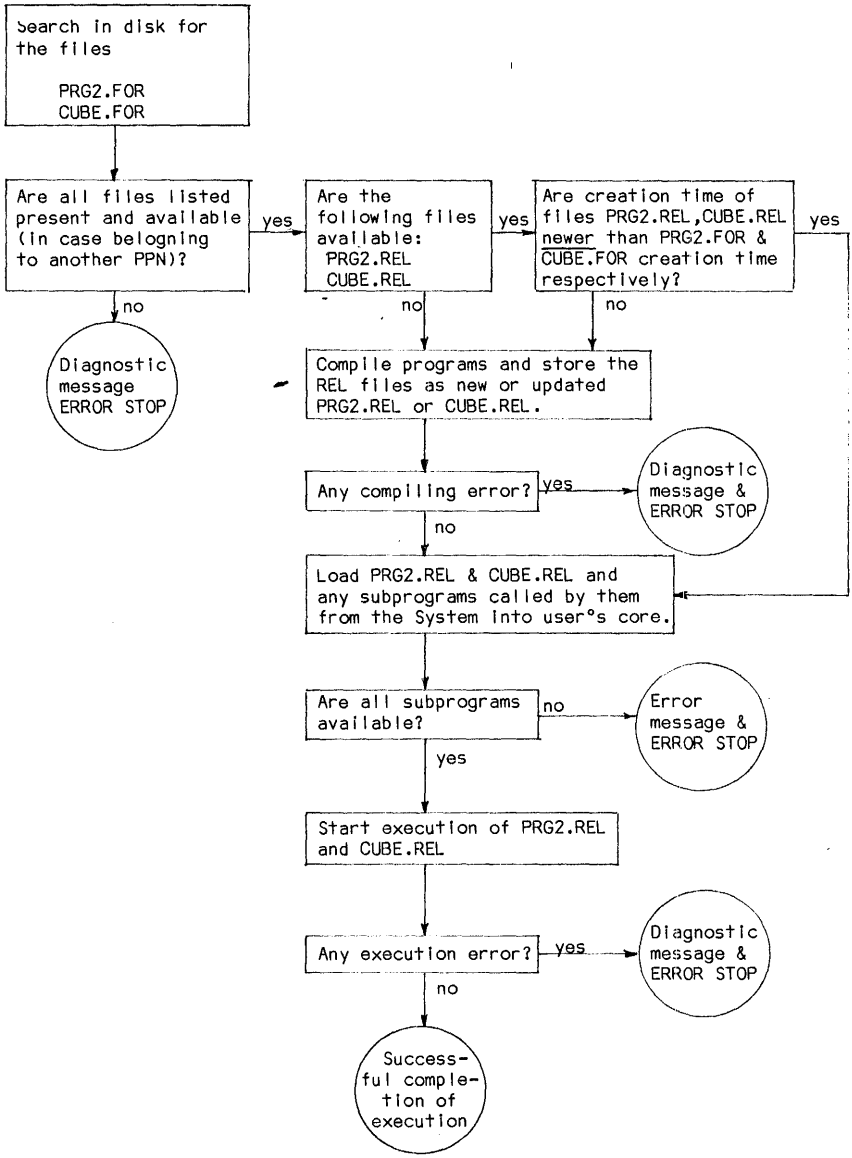


Figure 3.1 Sequence of Operations for "EXECUTE PRG2.FOR,CUBE.FOR"

```
.
FORTRAN 5A(621): PRG1.FOR
MAIN.   OCTAL PROG SIZE=43
LINK:   Loading
[LNKXCT PRG1 execution]

        1         1
        2         8
        3        27
        4        64
        5       125
        6       216
        7       343
        8       512
        9       729
       10      1000

End of execution    FOROTS 5B(1001)
CPU time: 0.08  Elapsed time: 1.05
EXIT
```

```
FORTRAN 5A(621): PRG2.FOR
MAIN.   OCTAL PROG SIZE=35
FORTRAN 5A(621): CUBE.FOR
CUBE    OCTAL PROG SIZE=52
LINK:   Loading
[LNKXCT execution]
>
>
        1         1
        2         8
        3        27
        4        64
        5       125
        6       216
        7       343

End of execution    FOROTS 58(1001)
CPU time: 0.05  Elapsed time: 7.50
EXIT
```

The three stages of compiling, loading and execution of a FORTRAN-10 program are carried out by a single EXECUTE command. These steps can also be carried out one at a time.

The monitor command *COMPILE list* will compile the FORTRAN files in the list and store the generated REL files, giving them the same filename but with an extension of REL.

The monitor command *LOAD list* will compile the programs, store the generated REL files, and also load them into the core.

The execution of the stored FORTRAN programs can also be accomplished by submitting the EXECUTE commands in cards. The following are two card assemblies for the batch jobs of executing Program 1 and Program 2:

```
$JOB [115103,320571]
$PASSWORD DEBBIE
.EXECUTE PRG1.FOR
$EOJ
```

```
$JOB[115103,320571]
$PASSWORD DEBBIE
.EXECUTE PRG2.FOR,CUBE.FOR
  1
  7
$EOJ
```

Once the compiling is done on a FORTRAN program, its object program is stored on the disk, and subsequent execution of the same program will bypass the compiling stage. In this manner, unnecessary compiling may be avoided. However, if the FORTRAN program belongs to another PPN, a user should not only ascertain if the FORTRAN program is protected against his access, but he should also determine whether he can gain access to a compiled REL file. If a REl file is already available and accessible, the command EXECUTE will directly access the REL files. In many cases, the source programs are proprietary, but the REL files are available for public access.

If a program will be used many times, a more efficient way of loading can be done in this way. After the program in the "list" of the "LOAD" command are loaded, the core content of the user's area in the core memory may be saved as a file with an EXE extension. The monitor commands to save a core image are LOAD

and SAVE as shown below:

```
.LOAD list
.SAVE NAME
```

and the saved file will have a name of NAME.EXE. Once that is done, subsequent execution of the program may be done by a command of:

```
.RUN NAME
```

where "NAME" is the the name of the specified EXE file.

This procedure is particularly advantageous if (1) a program will be used repeatedly, or (2) the list of programs in the EXECUTE command contains many files and many file specifications. Some of the files may reside on slow and busy peripherals such as the DECtape.

3.4 Optional Switches

The monitor command EXECUTE requires the use of three service programs: the monitor, the FORTRAN compiler, and the loader. In each of the three processors, options are implemented to allow a user to select some variation of services. These options are called switches. Switches are available on all three service processors, and they are separately discussed next.

(1) Monitor switches The details of the switches for the command COMPILE, LOAD and EXECUTE will be given in Chapter 8, so only the most frequently used switches are listed below. The monitor switch has a form of a slash followed immediately by a word which can be abbreviated. These switches and their functions are listed in Table 3.1.

(2) Compiler switches While the monitor program is somewhat uniform among the DEC System-10 users, the compilers--particularly the FORTRAN compiler-- may have many versions, and some with local modifications. Selected switches which appear on the same command line as those of the compiler switches are words enclosed in parentheses. These switches are listed in Table 3.2.

(3) Loader switch The format of a loader switch is a percent sign (%) followed by one or two characters. Three such switches are listed in Table 3.3.

Example: *.EXECUTE SAMPLE.FOR/LIST*
Function: Compile SAMPLE.FOR, store SAMPLE.REL on disk, load it into the core, and execute. Also, generate a source listing file SAMPLE.LST.

Example: *.EXECUTE SAMPLE.FOR/CREF (I) %OM*
Function: Compile (including all D-statements), load and execute. Generate a cross reference file for later CREF program, and produce a loader map at the terminal.

Monitor Switch	Function
/COMPILE	To force a compiling even if there already exists a REL file. The purpose of this switch is to force the use of compiler because certain compiler switches are also chosen in the EXECUTE command. Otherwise, the compiler is bypassed if there already exists a valid REL file bearing the same filename.
/CREF	To produce a cross-reference listing file on the disk for each file compiled for later processing by the CREF program. The cross-references include such information as variable names, statement labels, and their cross references. Before the user signs off, he may get a printout copy of the cross-reference by another monitor command: CREF. If the CRF file generated during a previous session at the terminal still is stored on disk, a list may be obtained by running the CREF program in the following ways: .R CREF *LPT:=NAME.CRF .R CREF *TTY:=NAME.CTF This will produce a copy of listing on the line printer (the left version) or on the terminal (the version on the right).
/LIST	To generate a disk listing file for each file compiled with the same filename, but with an extension of LST. These files can be listed with the PRINT or QUEUE command (see Chapter 8). If a REK file already exists, this switch will be ignored unless a forced compiling is ordered by the /COMPILE switch.
/LIBRARY	To select the loading of only those subroutines and functions referenced in the programs. Otherwise, the entire library file will be loaded.
/DEBUF:BOUNDS	To report if subscripts get out of bounds as defined by the DIMENSION statement for that array. This is one of the most common errors.

Table 3.1 Selected Monitor Switches

Example: *.EXECUTE SAMPLE.FOR, PRG:IMSL/LIBRARY*
Function: Compile the source program SAMPLE.FOR and thus generate SAMPLE.REL. Then load it along with those subroutines in PRG:IMSL that are called by the program SAMPLE.FOR. The LIBRARY switch here is absolutely necessary because the package PRG:IMSL contains about 400 subroutines. Execute when loading is completed.

Compiler Switch	Function
(INCLUDE) or (I)	To compile the program by regarding all statements with "D" in column 1 as FORTRAN statements. If this switch is not specified, those statements will be regarded as comments and bypassed. The frequent uses of this switch is to insert the debugging statement as the "D-statements," which are usually output statements to type out intermediate results or to type out tracing progress, such as a message "Reaching check point 5." Once a program is completely debugged, it can be compiled again, but this time without the INCLUDE-switch, and all D-statements will be ignored.
(NOERROR) or (NOE)	To suppress error message on user's terminal. The error message will only appear on the listing file if it is requested by the /LIST or the /CREF switch.
(NOWARNINGS) or (NOW)	To suppress warning messages on the terminal.
(OPTIMIZE) or (O)	To perform global optimization of compiling.

Table 3.2 Selected FORTRAN Compiler Switches

Loader Switch	Function
/S	To load local symbols used primarily for debugging purpose along with the program.
/1M	To type out a loader map at the user's terminal and include local symbols. In a batch job, the loader map with this switch will be included in the log file.
/0M	To type out a loader map at the user°s terminal. In a batch job, this switch will include the loader map in the log file.

Table 3.3 Selected Loader Switches

3.5 An Example of FORTRAN Processing

As an illustration of FORTRAN-10 programming and processing on a time-sharing system, an example will be carried through in all steps. The problem deals with the solution of an equation $Ax + Bx + Cx + D = 0$ with significance to 3 digits. The FORTRAN program for the problem is listed below:

```
C     SAMPLE PROBLEM FOR FORTRAN-10
      READ(5,10)A,B,C,D,X1
   10 FORMAT(F20.7)
    1 X2=X1-(A*X1**3+B*X1**2+C*X1+D)/(3.*A*X1**2+2.*B*X1+C)
      WRITE(6,10)X2
      IF(ABS((X1-X2)/X2)-0.001)3,3,2
    2 X1=X2
      GO TO 1
    3 WRITE(6,11)X2
   11 FORMAT(/' THE REAL ROOT = ', F20.7)
      STOP
      END
```

The rest of this section shows a case history of running this problem, from entering the program, through debugging and editing and finally executing it. Written running comments were added to aid understanding. All text in italics represent the user's own typing; all others are the computer's printout.

```
.UPDATE NEWTON.FOR
[CREATING NEW FILE]
XC     SAMPLE PROGRAM FOR FORTRAN-10
>      READ(5,10)A,B,C,D,X1
>   10 FORMAT(5F)
>    1 X2=X1-(A*X1**3+B*X1**2+C*X1+D)/(3.*A*X1**2+2.*B*X1+C)
>      WRITE(6,19)X2
>      IF(ABS((X1-X2)/X2 .001)3,3,2
>    2 X1=X2
>      GO TO 1
>    3 WRITE(6,11)X2
>   11 FORMAT(/' THE REAL ROOT = ', F20.7)
>     STOP
>      END
>$END
1 blocks written on NEWTON.FOR[115103,320571]
```

Enter a new program by UPDATE

```
.EXECUTE NEWTON.FOR
FORTRAN 5A(621): NEWTON.FOR
00006            IF(ABS((X1-X2)/X2 .001,3,3,2
?FTNUMP LINE:00006 UNMATCHED PARENTHESES
00010       11 FORMAT(/' THE REAL ROOT = ', F20.7)
00011        STOP
?FTNFWE LINE:00010 FOUND "T" WHEN EXPECTING A END OF STATEMENT

UNDEFINED LABELS

   19       11

?FTNFTL   MAIN.            4 FATAL ERRORS AND NO WARNINGS
LINK:   LOADING
[LNKNSA No start address]
EXIT
```

Missing minus sign

Use error diagnosis message to help with debugging.

```
.UPDATE NEWTON.FOR
C      SAMPLE PROGRAM FOR FORTRAN-10
>$TO 6
       IF(ABS(ABS((X1-X2)/X2.001)3,3,2
>$CHANGE/ .001/)-.001/
       IF(ABS((X1-X2)/X2)-.001)3,3,2
>$TO/STOP/
       STOP
>$CHANGE/STOP/ STOP/
       STOP
>$END
1 blocks written on NEWTON.FOR[115103,320571]

EXIT
.EXECUTE NEWTON.FOR
FORTRAN 5A(621): NEWTON.FOR

UNDEFINED LABELS

    19      11

?FTNFTL   MAIN.         4 FATAL ERRORS AND NO WARNINGS
LINK:   Loading
[LNKNSA No start address]

EXIT


.UPDATE NEWTON.FOR
C      SAMPLE PROGRAM FOR FORTRAN-10
>$TRAVEL/19/
       WRITE(6,19)X2
>$CHANGE/19/10/
       WRITE(6,10)X2
>$GO
?>$END
1 blocks written on NEWTON.FOR[115103,320571]

EXIT


.EXECUTE NEWTON.FOR
FORTRAN 5A(621): NEWTON.FOR
MAIN.   OCTAL PROG SIZE=145
LINK:   Loading
[LNKXCT NEWTON execution]
1.0  -16.0  65.0  -50.0  16.0
    12.9158000
    11.1082200
    10.2498400
    10.1173400
    10.0000900
    10.0000000

THE REAL ROOT =          10.0000000
STOP

End of execution   FOROTS 5B(1001)
CPU time: 0.09  Elapsed time: 23.98
EXIT
```

Make changes by UPDATE

Error still exists.

Search for statement 19

Search for more "19".

"?" says "can't find any more."

Execute again

Compile and load successfully

Input data for
$x^3 - 16x^2 + 65x - 50 = 0$
with initial trial value $X1 = 16$

Answer: $x = 10$

A SUMMARY OF FORTRAN-10

This part of the chapter is devoted to a summary of the FORTRAN-10 language, which is an enhancement of the ANSI standard FORTRAN. The enhancement may be a new FORTRAN statement, such as the IMPLICIT-statement; or it may be some additional features in a standard FORTRAN statement, such as those in the DIMENSION-statement. These enhancements will be identified in the summary by a heavy vertical line on the left side of the page, for example:

(5) A debug line A debug line has a character "D" or "d" etc etc etc

The identification of the enhancement will be useful in the conversion of a FORTRAN-10 program to other versions of FORTRAN, or vice versa.

3.6 A Summary of Constants, Variables and Expressions

(1) Constants There are nine types of constants in FORTRAN-10: integer constants, real constants, double precision constants, complex constants, logical constants, literal constants, octal constants, double octal constants, and statement label constants, as summarized in Table 3.4:

Constant	General Form	Remarks and Examples
Integer constant	no decimal point	ranging from $-2^{35}+1$ to $2^{35}-1$
Real constant	always with a decimal	7 to 9-digit precision in mantissa
Double precision constant	exponent symbol is D	3.00D2=300.0000000000000 (16-digit precision)
Octal constant	signed or unsigned octal preceded by a '"'	"567, "-567
Double octal constant	same as single precision octal	"1234567000123456700
Complex constant	(x,y)	(3.1,-4.7) for 3.1-j4.7
Logical constant	.TRUE. .FALSE. "-1 "0	
Literal constant	'QUOTE' nHxxxxx	'TIME' 4HTIME
Statement label	1 to 5 decimal digits preceded by "$" or "&"	$1234 &999

Table 3.4 A Summary of FORTRAN-10 Constants

(2) Variables Variables are specified by names and types. The name of a variable consists of one to six alphanumeric characters, the first of which must be alphabetic. The type of a variable may be specified explicitly by a type declaration statement or implicitly by the IMPLICIT statement. If the variable is not specified in this manner, then a first letter of I, J, K, L, M or N indicates an integer variable; any other first letter indicates a real variable.

Variable arrays carry subscripts that are integer constants, variables or expressions. In addition, the following are permitted in FORTRAN-10:

A. A subscript may contain a non-integer arithmetic expression. However, when such a subscript is evaluated, it is truncated and converted to an integer after its evaluation.

B. A subscript may contain a function reference such as A(10*SIN(X)).

C. Subscripted variables may be used as subscripts or nested subscripts of subscripted variables.

(3) Expressions Compounded numeric expressions must be constructed according to the following rule. With respect to the numeric operators of +, -, *, /, any type of quantity (integer, real, double precision, complex, logical, literal, octal or statement label) may be operated with any other, with one exception: A complex quantity may not be operated with a double precision quantity. The result of these mixed mode operations are tabulated in Table 3.5. (Mixed mode operations are not allowed in ANSI FORTRAN.)

Operation		Type of Argument 2				
+, -, *, /		Integer	Real	Double Precision	Complex	Others
Type of Argument 1	Integer	Integer	Real	Double Precision	complex	Integer
	Real	Real	Real	Double Precision	Complex	Real
	Double Precision	Double Precision	Double Precision	Double Precision	Not Allowed	Double Precision
	Complex	Complex	Complex	Not Allowed	Complex	Complex
	All Others	Integer	Real	Double Precision	Complex	Octal

Table 3.5 Results of Mixed Mode Operations

For example, if X is real in an expression (3.1,-4.1)*X, the expression will be complex after evaluation.

The logical operators and relational operators are listed in Table 3.6 and Table 3.7 respectively.

Logical Operators	Meaning	Example
.NOT.	Negation	.NOT.P
.AND.	$\cap$	P.AND.Q
.OR.	$\cup$	P.OR. Q
.XOR.	$\oplus$	P.XOR.Q
.EQV.	$\odot$	P.EQV.Q

Table 3.6 Logical Operators

Relational Operators	Meaning
.GT.	$>$
.GE.	$\geq$
.LT	$<$
.LE.	$\leq$
.EQ.	$=$
.NE.	$\neq$

Table 3.7 Relational Operators

A summary of FORTRAN-10 library functions is shown on Table 3.8.

3.7 FORTRAN-10 Statements

The field format of a FORTRAN-10 statement follows the general rules of FORTRAN-IV statement. There are certain differences associated with a FORTRAN-10 line. In FORTRAN-10, there are following different types of statement lines:

(1) An initial line If a FORTRAN-10 statement has continuation lines, the first line of the group is called an initial line.

(2) A continuation line A continuation line is identified by any character (except for a blank or zero) placed in column 6. A maximum of 20 lines are permitted in a FORTRAN-10 statement including the initial line. Continuation lines may not be interrupted by comment lines.

(3) A multi-statement line A multi-statement line combines several successive statements in a single statement, each component separated from the other by a semicolon (;). If the multi-statement carries a statement number, it is always associated with the first component. For example, two separate statements:

```
A = B*C
X = Y+Z
```

can be combined into a single line as: A = B*C; X = Y+Z

(4) A Comment line A comment line has one of the characters (C,$,/,*,!) placed in column 1. Comments may also be added to any statement in the field of columns 7-72, provided that a character (!) precedes the text. For example:

```
A = B*C ; X = Y+Z  !STEP NO. 1
```

(5) A debug line A debug line has a character "D" or "d" in column 1. When the program is compiled, it is ignored unless there is an "(INCLUDE)" switch in the command. This is used for debugging purposes, such as an output line for tracing.

Function	Form	Definition	Type of Argument	Type of Result
Absolute values:				
Real	ABS	|arg|	Real	Real
Integer	IABS		Integer	Integer
Double	DABS		Double	Double
Complex to real	CABS	$c = \sqrt{x^2 + y^2}$	Complex	Real
Conversion:				
Integer/real	FLOAT	Float(Arg)	Integer	Real
Real/integer	IFIX	Integer(arg)	Real	Integer
Real(cmplx)	REAL	REAL part(cmplx arg)	Complex	Real
Imag(cmplx)	AIMAG	IMAG part(cmplx arg)	Complex	Real
Real/Cmplx	CMPLX	c=Arg1 + j Arg2	2 Reals	Complex
Cmpx conjugate	CONJG	c=conjugate(cmplx arg)	Complex	Complex
Truncation:				
Real/real	AINT	Real truncation	Real	Real
Real/integer	INT	Integer truncation	Real	Integer
Remaindering:				
Real	AMOD	Remainder(arg1/arg2)	2 Reals	Real
Integer	MOD	Remainder(arg1/arg2)	2 Integers	Integer
Square root:				
Real	SQRT	$\sqrt{arg}$	Real	Real
Double	DSQRT		Double	Double
Complex	CSQRT		Complex	Complex
Logarithm:				
Real	ALOG	Ln (arg)	Real	Real
	ALOG10	Log (arg)	Real	Real
Double	DLOG	Ln (arg)	Double	Double
	DLOG10	Log (arg)	Double	Double
Complex	CLOG	Ln (arg)	Complex	Complex
Sine:				
Real (radians)	SIN		Real	Real
Real (degrees)	SIND		Real	Real
Double (radians)	DSIN	sin (arg)	Double	Double
Complex	CSIN		Complex	Complex
Cosine:				
Real (radians)	COS		Real	Real
Real (degrees)	COSD		Real	Real
Double (radians)	DCOS	cos (arg)	Double	Double
Complex	CCOS		Complex	Complex
Arc sine	ASIN	$\sin^{-1}$(arg)	Real	Real
Arc cosine	ACOS	$\cos^{-1}$(arg)	Real	Real
Arc tangent:				
Real	ATAN	$\tan^{-1}$(arg)	Real	Real
Double	DATAN	$\tan^{-1}$(arg)	Double	Double
Two real arg	ATAN2	$\tan^{-1}$(arg1/arg2)	Real	Real
Exponential:				
Real	EXP	$e^{(arg)}$	Real	Real
Double	DEXP		Double	Double
Complex	CEXP		Complex	Complex
Hyperbolic:				
Sine	SINH	sinh (arg)	Real	Real
Cosine	COSH	cosh (arg)	Real	Real
Tangent	TANH	tanh (arg)	Real	Real
Maximum value:				
Real	AMAX1	Max(a1,a2,...)	Reals	Real
Integer	MAX0	Max(k1,k2,...)	Integers	Integer
Minimum value:				
Real	AMIN1	Min(a1,a2,...)	Reals	Real
Integer	MIN0	Min(k1,k2,...)	Integers	Integer
Random number	RAN	random number between 0 and 1	dummy	Real

Table 3.8 FORTRAN-10 Library Functions

(6) A blank line This is ignored in compiling, but useful in making the listing easier to read.

Various types of FORTRAN-10 statements will now be discussed. As in all versions of the FORTRAN language, the order of the FORTRAN-10 statements is important in a program. The proper order of the statements is summarized in Table 3.9.

<table>
<tr><td rowspan="6">COMMENT</td><td colspan="3">PROGRAM. FUNCTION, SUBPROGRAM
or BLOCK DATA statements</td></tr>
<tr><td rowspan="5">FORMAT statements</td><td colspan="2">IMPLICIT statements</td></tr>
<tr><td colspan="2">PARAMENTER statemets</td></tr>
<tr><td colspan="2">DIMENSION, COMMON,
EQUIVALENCE, EXTERNAL
NAMELIST, or TYPE
Specification
statements</td></tr>
<tr><td rowspan="2">DATA
statements</td><td>Statement
function
Definitions</td></tr>
<tr><td>Executable
statements</td></tr>
<tr><td colspan="3">END statement</td></tr>
</table>

Table 3.9 A Summary of FORTRAN-10 Statement Sequence

The list of statements in each box indicates the order in which these statements must appear. The table also indicates that certain statements may be placed anywhere in the range shown in the Table. For example, a FORMAT statement may be placed anywhere after the PROGRAM statement and before the END statement.

3.8 A Summary of FORTRAN-10 Compilation Control Statements

Statement	Function
PROGRAM *name*	This statement instructs the compiler to assign "*name*" instead of MAIN as the name of a program. "*name*" must be 6 characters or less. This statement, if written, must be the first statement of a program.
INCLUDE '*file*'	*file*= standard file specification. This statement allows an inclusion of a code segment in a program unit.
END	Physically the last statement of a program or a subprogram.

Table 3.10 A Summary of FORTRAN-10 Compilation Control Statements

3.9 A Summary of Specification Statements

The specification statements specify the type characteristics, storage allocations, and data arrangements. They are summarized in Table 3.11:

Statement	Function
DIMENSION *S1,S2,...Sn*	where *Si* is an array declarator of either of two form: *VARIABLE(max1,max2,...,maxn)* *VARIABLE(min1:max1, min2:max2,..., minn:maxn)* and "mini:max" value represents the lower and upper bounds of an array dimension. The symbol colon (:) may be replaced by a slash (/) as a delimiter. When used in a subprogram, the array dimension may be an integer constant or an integer variable, thus making the dimension adjustable in a subprogram.
TYPE *list*	where TYPE may be one of the following: INTEGER, REAL, DOUBLE PRECISION, COMPLEX or LOGICAL. Size modifiers are acceptable in FORTRAN-10 but are interpreted differently: type*1 = acceptable but interpreted as a full word type*2 = full word type*4 = full word type*8 = double precision
IMPLICIT *TYPE(a1,A2..) TYPE(b1,B2,..),...*	where A1,A2,...,B1,B2,... are letters. This statement declares the data type of variables and functions according to the first letters. A range of letters may be specified by a dash between the first and the last letters, for example: IMPLICIT INTEGER (A-N)
COMMON */block indentifier/identifier,identifier,...identifier*	The COMMON statement causes specified variables or arrays to be stored in an area available to other programs. By means of COMMON statements, the data of a main program and/or its subprograms may share a common storage area.
EQUIVALENCE $(V_1,V_2,...)$, $(V_1,V_{k+1},...)$, ...	The EQUIVALENCE statement causes more than one variable within a given program to share the same storage area.
EXTERNAL *name1, name2, ...*	Distinguish the names as names of subprograms to be used as arguments to other subprograms.
PARAMETER *P1=C1, P2=C2, ...*	where Pi = a standard user-defined identifier, Ci = any type of constant This statement defines constants symbolically during compilation.
DATA $list/d_1,d_2,.../$, $list2/d_k,d_{k+1},.../$, ...	The data to be compiled into the object program is specified in this statement. The "list" may be a full array or an partial array in an implied DO format.

Table 3.11 A Summary of Specification Statements

3.10 A Summary of Assignment Statements

The assignment statements are summarized in Table 3.12:

<table>
<tr><th>Statement</th><th>Function</th></tr>
<tr><td>VARIABLE = EXPRESSION</td><td>The EXPRESSION in an assignment statement may be an arithmetic or a logical expression. Their formats are the same. In an arithmetic expression, mixed mode is permitted in FORTRAN-10. The rules of mixed mode expression results depend on the type of VARIABLE in the statement. Note that we are dealing with FORTRAN statements here, while a previous Table 3.5 lists the results of mixed mode operations in a sub-expression. The rules are now summarized below:

Mixed Mode Statement

<table>
<tr><th>Expression Type</th><th colspan="5">Variable Type</th></tr>
<tr><th></th><th>Real</th><th>Integer</th><th>Complex</th><th>Double</th><th>Logical</th></tr>
<tr><td>Real</td><td>D</td><td>C</td><td>R,I</td><td>H,L</td><td>D</td></tr>
<tr><td>Integer</td><td>C</td><td>D</td><td>R,C,I</td><td>H,C,L</td><td>D</td></tr>
<tr><td>Complex</td><td>R</td><td>C,R</td><td>D</td><td>-</td><td>R</td></tr>
<tr><td>Double</td><td>H</td><td>C,H,L</td><td>-</td><td>D</td><td>H</td></tr>
<tr><td>Logical</td><td>D</td><td>D</td><td>R,I</td><td>H,L</td><td>D,H</td></tr>
<tr><td>Literal</td><td>D,H %</td><td>C,H %</td><td>D &</td><td>D &</td><td>D %</td></tr>
</table>
Legend: D = direct replacement
C = conversion with truncation
R = real part only
I = imaginary part set to 0
H = high order only
L = low order part set to 0

Note: % = use of the first part of the literal
& = use the first two words of the literal</td></tr>
<tr><td>ASSIGN n TO I</td><td>This is used to assign a statement label constant to a variable name, which will become a statement label variable.</td></tr>
</table>

Table 3.12 A Summary of Assignment Statements

3.11 A Summary of Control Statements (Table 3.13)

Statement	Function
GO TO *n*	An unconditional transfer statement
GO TO (*n1*,*n2*,...,*nk*) or GO TO (*n1*,*n2*,...,*nk*)	Assigned GO TO statement
GO TO *k* OR	GO TO *k*, (*L1*,*L2*,...*Ln*) Assign GO TO statement
IF (*E*) *L1*,*L2*,*L3*	Conventional arithmetic IF statement where *E* = an arithmetic expresssion
IF (*E*) *S*	where S is an executable statement. This is a conventional logical IF statement, where E is a logical expression.
IF (*E*) *n1*,*n2*	where *n1* and *n2* are two statement labels. This is a two-exit logical IF statement and *E* = a logical expression. This statement will transfer the execution to statement label *n1* if *E* equals .TRUE., and to statement *n2* if *E* = .FALSE. In other words, this is an "IF-THEN, OTHERWISE" statement.
DO *n I* = *m1*, *m2*, *m3* where *n* = terminal statement label *I* = index variable *m1* = initial parameter *m2* = terminal parameter *m3* = increment parameter Note: (1) Nested DO's follow conventional rules. (2) Index variable should not be altered within the loop range. Even an inclusion as a subprogram argument may produce a warning message during compiling. (3) The index variable may be an integer or a real variable. The parameters may be integer or real expressions, which will be calculated at the beginning of the DO loops. (4) Real, integer, positive, negative, zero constants are all permitted for *m1*,*m2*,*m3*. Thus the FORTRAN-10 DO-statements allow decrements, negative indices, non-integer numeric indices.	
STOP , or STOP '*literal string*' , or STOP *n*	Terminal will print the *literal string* as a message or *n* as a message.
PAUSE, or PAUSE *literal string* , or PAUSE *n*	The PAUSE statemnt will cause the following message to be printed at the terminal: TYPE G TO CONTINUE, X TO EXIT, T TO TRACE

Table 3.13 A Summary of Control Statements

3.12 Terminology Used in FORTRAN-10 INPUT/OUTPUT (I/O) Statements

One powerful feature of FORTRAN-10 is that it possesses a set of extremely powerful input/output statements, far more powerful than the standard set in the 1966 ANSI standard. In order to present the I/O statements, we will first get acquainted with some terminology:

(1) Transfer mode Data transfer between storage and I/O devices or between storage locations is done in several different modes:

a. Sequenctial mode This is the most common mode, in which the records are accessed or transferred in a sequential order immediately following the last accessed or transferrecd record.

b. Random access mode This permits the access and transfer of records from a file in any desired order. The OPEN (see Section 3.16) statement is required to establish an I/O mode of this kind.

c. Append mode This is a variation of the sequential mode. It permits writing a record immediately after the last record of the accessed file. The OPEN statement is required to establish an I/O mode of this kind.

d. Dump mode

(2) Keywords of I/O statements (Table 3.14)

Keyword	Transfer of Data
READ	from a peripheral device to the processor storage
REREAD	repeat the last READ
ACCEPT	from a terminal to storage
FIND	to locate the next recoprd to be read during a random access READ operation
DECODE	from a specified storage area into the
WRITE	from storage to a peripheral device
PRINT	from storage to a printer
PUNCH	from storage to a card punch
TYPE	from storage to a terminal
ENCODE	to transfer from the variables of a specified I/O list into a specified storage area

Table 3.14 A Summary of Keywords of FORTRAN-10 I/O Statements

(3) Basic formats and components of READ and WRITE statements

Basic Statement Form	Function
KEYWORD (u,f) list	Formatted I/o transfer
KEYWORD (u#R,f) list	Random access formatted I/O transfer
KEYWORD (u,) list*	Listed-directed I/O transfer
KEYWORD (u,name)	NAMELIST-controlled I/O transfer
KEYWORD (u) list	Binary I/O transfer
KEYWORD (u#R) list	Random access binary I/O transfer

where:

KEYWORD = READ or WRITE
u = logical unit number
f = format statement number
list = I/O list
#R = the delimiter # followed by the number of a record in an established (by an OPEN statement) random access file
* = symbol specifying a list-directed I/O transfer
name = the name of an I/O list defined by a NAMELIST statement

In additon, when a unit *u* is specified, the optional argument

ERR=e and *END=d*

may be added to any of the READ or WRITE statement.

Table 3.15 A Summary of READ/WRITE Basic Formats

(4) Logical unit number (Table 3.16)

Unit Number xx	Default Filenames	Use	
		Time-sharing	Batch
1 - 4	FORxx.DAT	DSK	DSK
5		TTY	CDR
6		TTY	LPT
7		CDP	CDP
8-30		DSK	DSK

Table 3.16 Logical Unit Number Assignments

These are decimal numbers to identify the physical devices used for most FORTRAN I/O operations. The devices should be explicitly specified in the OPEN

statement. The definitions of these unit numbers as well as how many are allowed are determined by the local installation. The typical DEC definition specifies units ranging from 1 to 63 assigned to the devices DSK, DECtapes, magtapes, CDR, LPT, PTR, PTP, etc. However, since a different system of peripheral device allocation is used at the University of Pittsburgh, the logical unit numbering system is revised and shown in Table 3.16. Installation at other institutions may have still diffeent definitions depending on the local configurations.

(5) Formatted and unformatted files Files transferred under the control of a format specification are called formatted files. Unformatted files are binary files transferred without a reference to a format specification and are transferred on an one-to-one correspondence between the source and the destination.

(6) Random access records The random access records are specified by an integer preceded by an apostrophe or a pound sign, for example, '123 or #123.

(7) List directed I/O The asterisk (*) is an I/O statement in place of a FORMAT statement number tells the compiler that the specified transfer operation is "list-directed." In a list-directed transfer, the data and their type are specified by the READ/WRITE I/O list. If a READ statement has an asterisk (*) where the FORMAT number usually is, the list-direct I/O will follow the rules listed below:

a. Octal constants in the list-directed I/O are not permitted.

b. Literal constants must be enclosed in single quotes, such as 'TIME'.

c. Blanks and commas are delimiters to separate different items in the I/O list.

d. Complex constants must be enclosed in parentheses.

e. If an item is inputted as a null (blanks, tabs, carriage returns, or linefeeds, but no data), the item will retain a previously inputted value.

f. A slash at any time will terminate the input operation even if the I/O list is not yet satisfied.

g. the repeat of a constant may be written as n*K, which means the constant K repeated n times.

(8) NAMELIST I/O lists The I/O lists are defined by a NAMELIST statement (see Section 3.17) in which each I/O list is named by a one- to six-character name that may be referenced by a READ/WRITE statement. I/O statements with a NAMELIST-defined I/O list cannot contain a FORMAT statement reference or a conventional I/O list. The only type of formatting permitted in the NAMELIST-controlled statements is an input record of $NAME var1=value1, var2=value2,...$.

3.13 A Summary of FORTRAN-10 READ Statements

Table 3.17 shows a summary of different types of FORTRAN-10 READ statements:

Statement	Function
Sequential Formatted READ:	
READ (u,f) list	This is the most frequently used form. It transfer data from logical unit *u* to storage.
READ (u,f)	Input data from unit *u* into either a H-field descriptor or a literal field descriptor given within the referenced format.
READ f	Same as *READ(u,f)* where =default unit for a card reader.
READ f, list	Read data from a card reader into storage.
Sequential Unformatted Binary READ:	
READ (u) list	Read one record from unit *u* into storage. The record must be previously prepared by a FORTRAN-10 unformatted WRITE statement.
Sequential List-Directed READ:	
READ (u,) list*	Read data from device unit *u* into storage as values of items in the list. If necessary, each item is converted to the type assigned in the list.
*READ *, list*	Read data from a card reader a list-directed list.
Sequential NAMELIST-Controlled READ:	
READ (u,name)	Read data from unit *u* into storage as the values of the items identified by the NAMELIST input specified by the name .
Random Access Formatted READ:	
READ(u#R,f) list	Input data from record *R* of unit *u* according to the referenced FORMAT *f*. The input files must be previously set up either by an OPEN or a DEFINE FILE command.
Random Access Unformatted READ:	
READ (u#R) list	Input data from record *R* of unit *u*. Place data into storage as values of items in the *list*. The input file must be a binary file prepared by a previously applied FORTRAN-10 unformatted random access WRITE statements.

Table 3.17 A Summary of FORTRAN-10 READ Statements

3.14 A Summary of FORTRAN-10 WRITE Statements

The WRITE statements resemble the READ statements in formats. Diffeent types of FORTRAN-10 WRITE statements are now summarized in Table 3.18:

Statement	Function
Sequential Formatted WRITE:	
WRITE (u,f) list	This is the most commonly used WRITE form. It transfers data from storage and outputs it on logical unit *u*.
WRITE (u,f)	Output the contents of any H-field or literal descriptor contained by to the logical unit *u*.
WRITE f	Same as *WRITE(u,f)* where *u*=default unit for a line printer.
WRITE f, list	Same as *WRITE(u,f)list* where *u*=default unit for a line printer.
Sequential Unformatted Binary WRITE:	
WRITE (u) list	Output the values of items in the *list* into the file associated with logcial unit *u*.
Sequential List-Directed WRITE:	
WRITE (u,) list*	Output data from storage into logical unit *u*.
Sequential NAMELIST-Controlled WRITE:	
WRITE (u,name)	Output data from storage into logical unit *u* with the values of items as identified by the NAMELIST-defined list specified by the name *name*.
Random Access Formatted WRITE:	
WRITE(u#R,f)list	Output into unit *u* the values from the storage identified by the contents of list to record *R*. Only the disk files that have been set up by either an OPEN statement or a call to the subroutine DEFINE FILE may be accessed by a WRITE statement of this form.
Random Access Unformatted WRITE:	
WRITE (u#R) list	Output into unit *u* the values from the storage identified by the contents of list to record *R*. Only the disk files that have been set up by either an OPEN statement or a call to the subroutine DEFINE FILE may be accessed by a WRITE statement of this form.

Table 3.18 A Summary of FORTRAN-10 WRITE Statements

3.15 A Summary of FORTRAN-10 I/O Statements

All FORTRAN-10 I/O statements, including the READ/WRITE statements already discussed are now summarized together in Table 3.19:

I/O Statement	Formatted	Transfer Format Control: Unformatted	Transfer Format Control: Namelist	List-Directed
READ Sequential	*READ(u,f)list* *READ f,list* *READ f*	*READ(u)list*	*READ(u,name)*	*READ(u,*)list* *READ *,list*
Random	*READ(u#R,f)list*	*READ(u#R)list*		
WRITE Sequential or, Append	*WRITE(u,f)list* *WRITE f,list* *WRITE f*	*WRITE(u)list*	*WRITE(u,name)*	*WRITE(u,*)list*
Random	*WRITE(u#R,f)list*			
REREAD Sequential	*REREAD f,list*			
FIND Random only	*FIND(u#R)*			
ACCEPT Sequential only	*ACCEPT f,list*			*ACCESPT *,list*
PRINT Sequential only	*PRINT f,list*			*PRINT *,list*
PUNCH Sequntial only	*PUNCH f,list* *PUNCH f*			*PUNCH *,list*
TYPE Sequential only	*TYPE f,list* *TYPE f*			*TYPE *,list*
ENCODE Sequtntial only	*ENCODE(c,f,s)list*			
DECODE Sequential	*DECODE(c,f,s)list*			

Legend:
u = logical unit number
f = format number
list = I/O list
name = name of specific NAMELIST I/O list
* = specify list-directed I/O
#R = local record position
c = number of characters per internal record
s = address of first storage

Table 3.19 A Summary of FORTRAN-10 I/O Statements

3.16 FORTRAN-10 File Control Statements

The FORTRAN-10 file control contains only two statements: OPEN and CLOSE. They are, however, among the most powerful and versatile statements in specifying the input/output files. The general forms are:

```
OPEN(arg1,arg2,...)
CLOSE(arg1,arg2,...)
```

The arguments have a genral form of *ITEM = value*. The power and versatility of the OPEN and the CLOSE statements are derived from the many options available as the arguments. These arguments are summarized and tabulated in Table 3.20(A&B).

Although there are many available options, many are special purpose type and not frequently used. The simplified version is just to take the most often used arguments: "unit", "file", "dispose" and "directory" in the OPEN statement, and just the "unit" in the CLOSE statement. Thus, the most often used forms are:

```
OPEN(UNIT=u,FILE='NAME.EXT',DISPOSE'value',DIRECTORY='m,n')

CLOSE(UNIT=u)
```

Example: `OPEN(UNIT=5, FILE='INPUT.DAT')`
Function: The disk file INPUT.DAT is opened on unit 5. If the FORTRAN program is written with unit 5 as the input unit, such as in the READ(5,f)list statement, the OPEN statement will change the program execution from TTY input to a file input. This is a convenient way of adapting an existing program from the TTY input to a disk file input.

Example: `OPEN(UNIT=1, FILE='INPUT.DAT', DIRECTORY='115103,320571')`
Function: The disk file IMPUT.DAT[115103,320571] is opened on unit 1.

Example:
```
OPEN(UNIT=3,ACCESS='SEQOUT',FILE='DATA.TMP')
  WRITE-statements on unit 3
CLOSE(UNIT=3)
OPEN(UNIT=1,ACCESS='SEQIN',FILE='DATA.TMP',DISPOSE='DELETE')
  READ-statements on unit 1
CLOSE(UNIT=1)
```
Function: An output file is opened on unit 3, to be named as DATA.TMP. The file is closed after output statge is completed; the file is reopened on unit 1 as an input file. The file is deleted from the disk when the CLOSE statement is executed.

Example:
```
OPEN(UNIT=1,FILE='INPUT.DAT',ACCESS='RANDOM',MODE='ASCII',
1    RECORD SIZE=80,PROTECTION="177)
```
Function: Open on unit 1 a disk file INPUT.DAT for random access I/O operation in ASCII mdoe. The records in the file are 80 characters long. When the CLOSE statement is executed, the file will be given a protection code of 177.

Argument	Possible Value	Function	Open*	Close*	Default Value
UNIT =	lv,lc	To define the logical unit number.	Req	Req	
DEVICE =	lv,lc	To specify the physical name or the logical name of an	Op	Op	logical name u
ACCESS =	Six possible values	To sepcify the type of input qnd/or output statements and the file access mode to be used in a specified I/O operations. The six possible values are: 'SEQIN' = to be read in sequential access mode 'SEQOUT' = to be written in sequential access mode 'SEQINOUT' = data file may be first read, then written record-by-record in a sequential access mode. At this access, a WRITE/READ sequence is illegal. 'RANDOM' = to specify random access mode in either READ or WRITE operation. The RECORD SIZE option is required when this access mode is specified. 'RANDIN' = to specify a read-only random access mode with a named file. 'APPEND' = to specify the APPEND mode. The record specified by an associated WRITE statement is to be added to the end of a named file. You must close it and then reopen the modified file to permit it to be read.	Op	ig	'SEQINOUT'
MODE =	four possible values	To define the character set of a file or record. Four possible values are: 'ASCII' = to specify an ASCII file 'BINARY' = to specify a FORTRAN formatted binary file 'IMAGE' = to specify an unformated binary file 'DUMP' = to specify the file to be handled in DUMP mode	Op	ig	'ASCII' for formatted file 'BINARY' for unformatted file
DISPOSE =	six possible values	To specify the action to be taken regarding a file at the close time. Six values are possible: 'SAVE' = to leave the file on the device 'DELETE' = to delete the file if it is on disk or on a DECtape. Otherwise, take no action. 'PRINT' = to queue the file for printing if it is a disk file. Otherwise, take no action. 'LIST' = to queue the file for printing and delete it if it is a disk file. Otherwise, take no action. 'PUNCH' = to output on paper tape punch. 'RENAME' = to change filename	Op	Op	'SAVE'
FILE =	lv,lc	To specify the name of the frile involved in the OPEN or CLOSE statement. The file name format is FLNAME.EXT. Default conditions: FLNAME = FLNAME.DAT FLNAME. = FLANME. (null) = FORxx.DAT where xx = two-digit unit number If the filenames of the same file in the OPEN and the CLOSE statements are different, the file is renamed.	Op	Op	'FORxx.DAT'
PROTECTION =	oc,lv	To specify a protection code. For example: PROTECTION = "155	Op	Op	"057

Table 3.20A FORTRAN-10 OPEN and CLOSE Statements

Argument	Possible Value*	Function	Open*	Close*	Default Value
DIRECTORY =		To specify the directory of the file. Most frequent use is to specify the PPN of the file. To specify a PPN of [123456,654321], use any of the three ways: (1) Single-precision array: *OPEN(unit=1,DIRECTORY=PATH,...)* where PATH and its elements are: *DIMENSION PATH(2)* *PATH(1)="123456* !project number *PATH(2)="654321* !programmer number (2) Double precision array: *OPEN(unit=1,DIRECTORY=PATH,...)* where PATH and its elements are: *DOUBLE PRECISION PATH(2)* *PATH(1)="000000123456000000654321* *PATH(2)="0* (3) Literal constants: *OPEN(unit=1,DIRECTORY='123456,654321',...)*	Op	Op	User's own PPN
BUFFER COUNT =	iv,ic	To specify the number of I/O buffers to be assigned to a particular device.	Op	ig	Monitor default value
FILE SIZE =	iv,ic	To specify disk file size in words	Op	ig	Monitor default
VERSION =	oc,iv	To specify the version number of the named file	Op	Op	0
BLOCK SIZE =	iv,ic	To specify block size for all storage media except disk and DECtape.	Op	ig	Monitor default
RECORD SIZE =	iv,ic	To specify record size in words. Required argument when specifying random access mode.	Op	ig	Monitor default value
ASSCIATE VARIABLE	iv	In random access mode, it provides storage for the number of the record to be accessed next if the program being executed were to continue to sequential access records starting from the current READ. For example, if record number 3 was read, the ASSOCIATE VARIABLE is 4.	Op	ig	
PARITY =	two possible values	To set the parity check system for magtape operation. Two possible values are 'ODD' and 'EVEN'.	Op	ig	System default value
DENSITY =	five values	To set the packing density of magtape. Five values are '200', '556', '800', '1600', and '6250'.	Op	ig	System default
DIALOG =	none iv,array	The use of this option in an OPEN statement enables you to supersede or defer, at execution time, the values previously assigned to the arguments of the statement. The System will return a message at the user's terminal: UNIT=n:/ACCESS=SEQINPUT/MODE=ASCII ENTER NEW FILE SPECS. END WITH AN ESC. Only the changed file specs needed be entered.	Op	ig	
ERR =	s	To go to statement No. s when there is an error during the execution of the OPEN or the CLOSE statement.	Op	Op	Error stop

*Legend: ic = integer constant; iv = integer variable;
lc = literal constant; lv = literal variable;
oc = octal constant,

Op = optional; ig = ignored.

TABLE 3.20B FORTRAN-10 OPEN and CLOSE Statements

Example:
```
OPEN(UNIT=1, FILE='INPUT.DAT')
   Other FORTRAN statements follow.
CLOSE(UNIT=1,FILE='OLD.DAT')
```
Function: Here we have the same unit number for the OPEN and the CLOSE statements, but they are different file name arguments. This is equivalent to renaming a file at the CLOSE time. The INPUT.DAT is renamed as OLD.DAT.

3.17 Format Statements

The FORMAT statements in FORTRAN-10 are in general compliance with the standard FORTRAN. Therefore, only a brief summary will be given here.

The FORMAT statement has a general form of

```
n FORMAT (S , S , ...)
```

where n is the statement number and each S is a data field specifier. The various data field specifiers are now summarized as follows:

(1) Numeric fields In the following list, "w" is an integer specifying the field width; "d" is an integer specifying the number of decimal places to the right of the decimal point or, for the G-format, the number of significant digits. For the D, E, F, and G inputs, the position of the decimal point in the external field takes precedence over the value of d in the format. This means that the decimal point of the input data need not be exactly at the specified column of the format. However, the data must be entered within the field specified in the format.

Floating-point type format	Fw.d
Exponent-type format	Ew.d
Double precision	Dw.d
General format:	
Real & double precision	Gw.d
Integer & logical	Gw
Complex	2Gw.d
Integer format	Iw
Octal format	Ow

(2) Numeric fields with scale factor Scale factors may be specified for D, E, F and G formats. A scale factor is written as nP where P is the identifying character and n is a signed or unsigned integer that specifies the scale factor.

For the F-type conversions (or G-type, if the external field is decimal fixed point), the scale factor specifies a power of ten so that:

$$\text{External number} = (\text{internal number}) * 10^{P}$$

For the D, E, and G (external field not decimal fixed point) formats, the scale factor multiplies the number by a power of ten, but the exponent is changed accordingly leaving the number unchanged except in form. For example, if the statement: FORMAT(F8.3,E16.5) is used to print out two values A and B:

the same numbers under a format of FORMAT(-1PF8.3,2PE16.5) would produce a printout of:

In input operations, the F-type data are the only type affected by the scale factor.

(3) Logical field The logical data field specifier is:

Lw

where "w" is an integer specifying the field width. If the format is used in an input operation, the first nonblank character in the data field is T or F, the value of the logical variable will be stored as TRUE or FALSE respectively. If the entire data field is blank or empty, a value of FALSE is stored. If the format is used in an output operation, (w-1) blanks followed by T or F will be output if the value of the logical variable is TRUE or FLASE respectively.

(4) Variable field width The numeric fields may appear in a FORMAT statement without the specification of the field width "w" or the number of places after the decimal point "d". When this format is used in an input operation, the input data can be entered in a "free form" style so long as a delimiter is used to separate two neighboring data. Any illegal character in a numeric field can be used as a delimiter. However, a good practice is to use either a comma (,) or a blank ( ) as a delimiter. For example, input according to the format:

```
10 FORMAT(2F,E,2I,D)
```

might appear as:

```
-2.34, 2.345, 0.5623E-01, 56, 783, 3.4567234569D+01
```

If such a format is used in an output operation, FORTRAN automatically assume the following field specifiers:

Format	Becomes
D	D25.16
E	E15.7
F	F15.7
G	G15.7 or G25.16
I	I15
O	O15

(5) Alphanumeric fields The format of an aphanumeric field is:

Aw or Rw

The maximum value of "w' is 5 for single precision, 10 for double precision. The A-field deals with variables containing left-justified, blank-filled characters; the R-field deals with variable containing right-justified, zero-filled characters.

(6) Alphanumeric data within a format statement Use nH format or enclose the alphanumeric data in single quotes. See examples below:

```
10 FORMAT(17H PROGRAM COMPLETE)
10 FORMAT(' PROGRAM COMPLETE')
```

(7) Complex field Complex quantitites are transmitted as two independent real quantities. The format specifier consists of two successive real specifiers or one real repeated specifier. For example, the following format can accommodate four complex quantitites:

```
10 FORMAT(4F10.4, 2E14.5, F10.5, F10.3)
```

(8) $ format descriptor A "$" format descriptor at the end of an output FORMAT is used to suppress the carriage return (and the associated line feed) at the end of the current record, except when the FORMAT is automatically repeated when the WRITE statement list contains more items than those in the FORMAT. One typical application is shown in the example below:

Example: The following is a segment of a FORTRAN-10 program:

```
10 FORMAT(' ANSWER YES OR NO '$)
11 FORMAT(A3)
   WRITE(6,10)
   READ(5,11)ANSWER
```

Function: When this segment of the program is executed, the following will appear on the user's terminal:

ANSWER YES OR NO > *(User answers YES or NO here)*

(9) Print control descriptor When FORTRAN output file is printed on a printer or a terminal, the first character of each line (or record) is reserved for the carriage control character which controls the spacing operations of the printer or the terminal. The FORMAT should have a beginning field of 1Ha where "a" is a desired control character. Table 3.21 lists the FORTRAN-10 print control characters.

3.18 FORTRAN-10 Device Control Statements

The FORTRAN-10 device control statements are normally used for magtape operation control, although they also work well with DECtapes and can be used to simulate disk devices. These tape control statements provide a set of run-time tape control instructions.

In order to execute these stantements, magtapes must first be MOUNTed, and a logical name of be given, where "u" is the logical unit for that tape unit in the FORTRAN program. Therefoe, if the device control statements are used in a FORTRAN-10 program, preliminaries such as the following must be carried out before the execution of the FORTRAN program*:

---

*Unless a run-time subroutine, such as RMOUNT, is available to mount a tape. See Section 3.21.

Print Control Character	ASCII Octal Value	Function
space	012	Skip to next line; skip to next page (form feed) after 60 lines.
0 zero	012	Skip a line
1 one	014	Form feed - go to top of next page
+ plus		Suppress skipping - overprint the line
* asterisk	023	@Skip to next line with no formfeed.
- minus	012	@Skip two lines.
2 two	020	@Space 1/2 of a page.
3 three	013	Space 1/3 of a page.
/ slash	024	@Space 1/6 of a page.
. period	022	@Triple space with a formfeed after every 20 lines printed.
, comma	021	@Double space with a formfeed after every 30 lines printed.

Table 3.21 FORTRAN-10 Print Control Characters
@=No effect on a terminal.

```
.DRIVE MT9
.MOUNT MT9:u/WE/VID:B313
```

Here, the VID used is for illustration. If there are more than one tape for the job, the above preliminaries must be done for every tape unit needed in the program.

The device control statements are now summarized in Table 3.22:

Statement	Function
REWIND u	Move and re-position the file back to the first record.
UNLOAD u	Rewind the source reel so that the tape is compiely off the take-up reel. The tape will be ready for unloading.
BACKSPACE u	Backspace one record except if it is already at record No.1. This statement cannot be used for files set up for random access,list-directed,or NAMELIST-controlled I/O operations.
ENDFILE u	Write an endfile record in the file located on device u.
SKIP RECORD u	Skip one record on device u.
SKIP FILE u	Skip one file which follows immediately the current one.
BACKFILE u	Backspace to the first record of the file preceding the current one.

Table 3.22 FORTRAN-10 Device Control Statements

3.19 FORTRAN-10 Subprogram Statements

Subprograms are procedures that are used repeatedly in a program or among the users, and therefore it is more convenient to define such common procedures so that they may be referenced. The arguments for such a common procedure are made general enough so that the subprograms can be utilized widely. These arguments are called dummy arguments. Dummy arguments in a FORTRAN-10 program may be one of the following: (1) variables, (2) array name, (3) subroutine identifiers, (4) function identifiers, or (5) statement label identifiers that are denoted by the symbol "*", "$", or "&".

These subprogram statements are now summarized in Table 3.23:

Statement	Function
NAME(arg1,arg2,...,argn) = expression	This defines an internal subprogram, where *NAME* is the name assigned, (*arg1,arg2,...*) is a list of dummy arguments.
TYPE FUNCTION NAME(arg1,arg2,...,argn)	where *TYPE* = optional type specification such as INTEGER, REAL, et. (*arg1,arg2,...*) = a list of dummy arguments.
SUBROUTINE NAME(arg1,arg2,...,argn)	
CALL NAME(arg1,arg2,...,argn)	Definition of a subroutine and calling a subroutine
ENTRY NAME(arg1,arg2,...,argn)	Multiple entry specification where: *NAME* = name to be assigned to the desired entry point. Rules of multiple entry in a FORTRAN-10 subroutien are given later.
RETURN	Return the control form the subroutine to the calling program. Next statement executed is one immediately following the calling statement in the calling program.
RETURN k	This is a multiple-return statement, where *k* is an integer constant, variable or expression. Rules of multiple return are given alter.

Table 3.23 A Summary of FORTRAN-10 Subprogram Statements

Often, many subprograms share a common computational procedure. Although these common procedures can again be made into subprograms to be called by subprograms, an alternative is to construct one subprogram with many entrance points. In Figure 3.2, a flow chart is shown for three entrance points and one exit. The entrance points are labeled as SUB (the front entrance), PTA and PTB (two side entrances). The program segments are represented as Segments 1, 2 and 3.

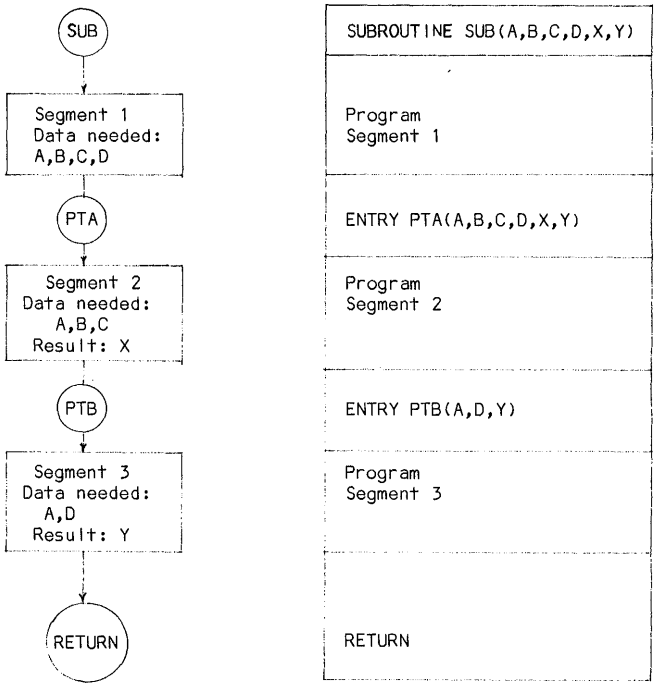


Figure 3.2 An Example of Multiple Entry Subprogram

The following rules on ENTRY should be noted:

(1) An ENTRY statemnet may not be placed in the main program.

(2) An ENTRY statement may not be placed in a DO loop.

(3) There is no need for the arguments of various ENTRY statements to agree with each other.

(4) Value of function must be returned by the use of current ENTRY name.

The statement *RETURN k* enables the selection of any labeled statement of the calling program as a return point. When the multiple returns form of this statement is executed, the assigned or calculated value of *k* specifies that the return is to be made to the kth statement label in the argument list of the calling statement. The value of k should be a positive integer that is equal to or less than the number of statement labels given in the argument list of the calling statement. If k is less than 1 or is larger than the number of available statement labels, a standard return operation is performed.

SUBPROGRAM LIBRARIES IN FORTRAN

3.20 Selected FORTRAN-10 Subprograms Developed by DEC (Table 3.24)

Subprogram Name	Effect
DATE(ARRAY)	"ARRAY" is a dimensioned variable in the calling pr5ogram with 2 elements. The suroutine will return the values: ARRAY(1) = 'DD-Mm', ARRAY(2) = 'm-YY' When ARRAY is printed with a 2A5 field format, the result is DD-Mmm-YY, for example, 19-Aug-80, the date when the subprogram was executed. To force the "month" part into all upper case letter, the following two statements should be inserted between the CALL DATE and WRITE statements: ARRAY(1) = ARRAY(1) .AND. "777777777677 ARRAY(2) = ARRAY(2) .AND. "577777777777 Then the above date example would be printed as 19-AUG-80.
TIME(X) or TIME(X,Y)	These subroutines will return a string constant X as 'HH:MM' as the current time in a 24-hour clock notation, and ' SS.S' for Y, where HH=hour, MM=minutes, SS.S=seconds.
ERRSET(N)	To control the typeout of execution-time arithmetic error messages. Message is suppressed after N occurences.
ERRSNS(I,J)	To determine the exact nature of an error on READ, WRITE, OPEN and CLOSE that was trapped with the "ERR=s" option in the statement. The subroutine will return two integers I,J. The (I,J) combination describes the nature of error according to a code table defined by DEC. (See Appendix H of Reference 4.)
EXIT	To terminate the subprogram.
RELEAS(u)	To release the logical unit u.
SAVRAN(I)	It sets its argument ot the last random number (interpreted as integer) that has been generated by the function RAN.
SETRAN(I)	The starting value of the function RAN is set to 1. If I=0, RAN uses its normal starting value.
SORT('OUTPUT=INPUT/switches')	The argument is a string representing a SORT prgram command. The details on the SORT program are given in Chapter 7. Check with local installation whether this subprogram is installed in the system.

Table 3.24 A Selection of FORTRAN-10 Subprograms Developed by DEC

3.21 Selected Subprograms Developed at the Pitt Computer Center

A group of subprograms have been developed and implemented in the FORTRAN-10 at the installation of the University of Pittsburgh. These subprograms are included for the convenience of Pitt users. DEC System-10 installation elsewhere would have similar types of subprograms but geared particularly to the local needs. These programs are often made available to other installations by exchange, lease or purchase. Since these suprograms have beeen implemented already in the Pitt FORTRAN-10, no additional monitor commands are needed to call them. For users elsewhere, they must confirm first with their installation personnel whether such or similar subprograms are available in their facilities.

The subprograms will be outlined according to their general functions:

(1) Supplementary library functions This group of subprograms are all functions and is used to supplement the DEC-supplied library functions (such as square root and sine function) which are given in Section 3.6 as Table 3.8. These supplementary functions are listed in Table 3.25:

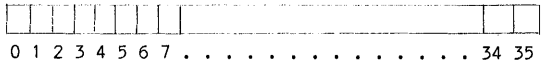
Function	Form	Definition	Type	
			Argument	Function
Tangent				
Real (radians)	TAN	tan(x)	real	real
Real (degrees)	TAND	tan(x)	real	real
Cotangent				
Real (radians)	COTAN	cot(x)	real	real
Real (degrees)	COTAND	cot(x)	real	real
Gamma function	GAMMA	(x)	real	real
Error function	ERF	erf(x)	real	real
Complementary erro function	ERFC	1 - erf(x)	real	real
CPU time	XEQTIM	CPU time in milliseconds	dummy	real

Table 3.25 Supplementary FORTRAN-10 Library Functions Developed at the University of Pittsburgh

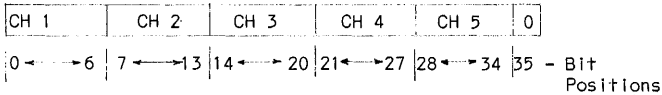
(2) Bit manipulation in a memory word

A DEC-10 memory word contains 36 bits. The hardware store a 37th bit for parity check, but that is of no concern to the user. The bits are numbered from 0 to 35 (from the most significant bit side to the least) as shown in Figure 3.3(a).

The group of bit-manipulation subprograms can be used for a wide range of applications, such as data re-formatting in data transfer between a magtape and disk storage. One particular application is in the area of character-storage manipulation. Since ASCII-coded characters are coded into 7-bit bytes, where a "byte" is a unit consisting any number of bits, each memory word can accommodate



(a) Bit Positions



(b) ASCII-Coded Character Storage

Figure 3.3 DEC-10 ASCII Storage Format

5 characters with one bit left over. The standard ASCII coded storage format is shown in Figure 3.3(b). As a result, bit-35 is always filled with a zero-bit when the word is an ASCII-coded word.

These subprograms are now outlined below:

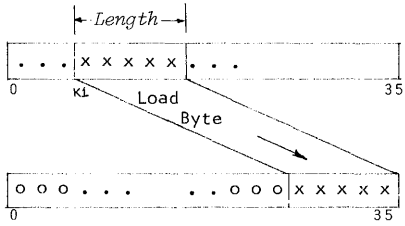
Function: *LDB(K1, Length, Z)*

- Source word to be processed
- Byte length in bits (integer, 1 to 36)
- Starting bit position (integer, 0 to 35)

Function: *LDBN(N, Length, Z)*

- Source word to be processed
- Bytesize in bits/byte (1 to 36)
- Byte number (from left to right)

Effect:



Subroutine: *CALL* *DPB (K1, Length, Z2, Z1)*

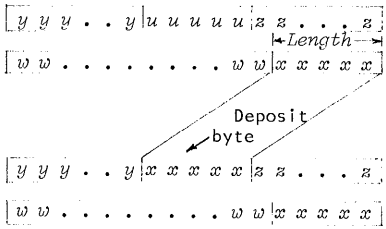
Starting bit position
Byte length
Destination word
Source word

Subroutine: *CALL* *DPBN (N, Length, Z2, Z1)*

Starting byte number
Bits/byte
Destination word
Source word

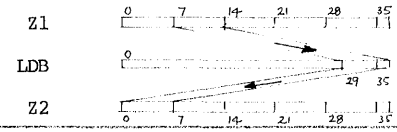
K1 and Length are integer constants or variables.

Effect "Before":



Z1 is unchanged. Z2 is unchanged except the deposited bute field.

Example: *CALL DPB(0,7,Z2,LDB(7,7,Z1))*
Exeution of this call will replace the first character in in the word Z2 by the second character in Z1 as shown below:



Function: *LSH (word, shift)*

number of places to be shifted:
+integer=shift left
-integer=shift right
word to be processed

Effect: *word:* y y y . . . y x x x x x

LSH(word,+k) y y···y x x x x x 0···0

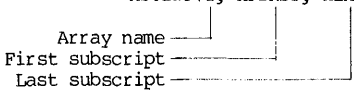
Example: *M = LSH('ABCDE',-14)*
The string word 'ABCDE' is shifted right 14 places. The shifted-out bits are replaced by 0's. Thus, the returned function value is M = ' ABC', because the ASCII-coded character for code 0000000 is null.

Subroutine: *CALL ZERO(ARRAY(I), ARRAY(J))*

Effect: Set all elements within the specified range to zero. Array may be of any type.

Example: *CALL ZERO(A(1), A(100))*
Set A(1), A(2), ...A(100) to 0.

Subroutine: *CALL ASCEND(Z, KFIRST, KLAST)*



Effect: Sort the Z-array from Z(KFIRST) to Z(KLAST) in an ascending order and then store them in the same array locations.

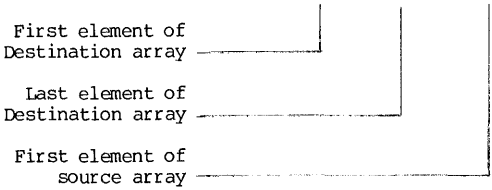
Example: *CALL ASCEND(X,1,100)*
Sort the X-array from X(1) to X(100) and store them in ascending order as the new X-array from X(1) to X(100).

Subroutine: *CALL SPRAY (Z(I), Z(J), VALUE)*

Effect: Set the Z-array of the specified subscript range to equal to the *VALUE*.

Example: *CALL SPRAY(Z1),Z(100),1.5)*
Set Z(1), Z(2),..., Z(100) to equal 1.5.

Subroutine: *CALL MOVE (A2(I), A2(J) , A1(K))*



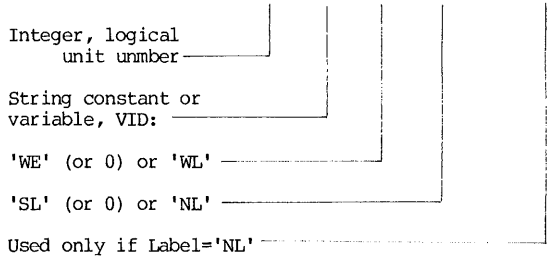
Effect: Copy an array A2 from A1 in this manner:

```
A2(I) = A1(K)
  .....
A2(J) = A1(K+J-1)
```

A1 and A2 arrays should be of the same type, and avoid double precision or complex array because the second word of each two-word element won't copy.

(4) Device and file specifications

Subroutine: *CALL RMOUNT (u , VID , WE , Label , Serial)*



Effect: A run-time MOUNT instruction for a magtape or DECtape.

Example: *CALL RMOUNT(1,'B313',0,0)*
This is equivalent to issuing two monitor commands before the execution of the FORTRAN program:
.DRIVES MT9
.MOUNT MT9:1/WE/VID:B313

Subroutine: *CALL IFILE (unit, filename, extension, PPN)*
CALL OFILE (unit, filename, extension, PPN)

where *unit* = integer constant, logical unit number
filename = 5-character or less string
extension =3-character or leess string
PPN = 12-digfit octal constant

Default extension is 'DAT'.
Default PPN is user's own PPN.

Effect: These are respectively equivalent to:

OPEN(unit=u,file='filename,extension',directory='p,pn',
access='seqin')
CLOSE(unit=u,file='filename,extension',directory='p,pn',
access='seqout')

Example: *CALL IFILE(1,'INPUT')*
Specify user's INPUT.DAT as an input file on unit 1.
CALL IFILE(2,'SAMPLE','TMP',"115103320571)
Note that although 6-character filename is given, IFILE and OFILE will only treat it as a maximum of 5-character string (because it is coded as ASCII instead of SIXBIT). Hence the search will be for a file SAMPL.TMPin the PPN of [115103,320571], instead of the specified file SAMPLE.TMP. If there is actually a file named SAMPL.TMP, this wrong file will be called. If there is no SAMPL.TMP, execution comes to an error stop.

3.22 The SUBSET Subprogram Package

Many subprograms have been developed by the faculty, staff and students at the University of Pittsburgh. Many of these are polished, optimized, and well documented. One such work is the SubSET (SUBprograms to Simplify Encoding Tasks), written by Ronal K. Nicholas* and stored under the PPN of [121403,250321]. By permission of Mr. Nicholas, a selection of SUBSET programs with their *subset* properties will be outlined. These subset properties are so chosen as to represent the salient points in these subprograms. For more details, the readers are referred to Reference 7, the SUBSET manual.

(1) Subprograms to report job information

Subprogram Name	Function or Subroutine	Effect
CORE(IP)	subroutine	Return an integer *IP* which is equal to the number of pages of core memeory for the current program with the fractions of page rounded to the next higher value.
IDENT(ID)	Function or subroutine	As a subroutine, it returns the argument *ID* as 15 ASCII characters in a 3-word array. The form of the ASCII string is '[m,n]' in three words. As a function, it also returns with "m" in the left half, and "n" in the right half of the returned word, both as 6-digit octal constants.
LOCATE(L)	Function or subroutine	As a subroutine, it sets the user's job to station *L*. If used as a function, it returns a functional value of .TRUE. if successful. Otherwise, it returns a value of .FALSE.
MYJOB(JOB)	Function or subroutine	Return a functional value or argument *JOB* the job number.
MYLINE(LINE)	Function or subroutine	It returns the argument *LINE* as the user's TTY line number. If it is a Batch job, the value is negative.
MYNAME(NAME)	subroutine	It returns a 3-word array containing 15 ASCII characters left-justified, which is the user's name as stored in the system.
WKDAY(TODAY)	subroutine	It returns a 3=character string which is the day of the day of the week, such as 'Mon', 'Tue', etc.

(2) Subprograms to manipulate arrays

These subprograms deal with initializing an array, copying one array onto another, and finding minimum and maximum elements in an array.

---

*Ronal K. Nicholas, Research Associate, Division of Research in Medical Education, School of Medicine, University of Pittsburgh

Subroutine: *CALL COPY (Z1, INCZ1, Z2, INCZ2, NTOTAL)*

- Z1 — First source array element
- INCZ1 — Z1-increment
- Z2 — First destination array element
- INCZ2 — Z2-increment
- NTOTAL — Total number of elements to be copied

Effect: Copy the Z1-array by the Z2-array with indicated starting array elements and subscript increments. Z1 and Z2 are the first array elements in the specified copying process. If the first element has a subscript of one, the subscript may be omitted, and Z1 or Z2 may appear as array name.

Example:

```
CALL COPY(X,1,Y,1,10)
This is equivalent to:      DO 5 I=1,10
                          5 Y(I)=X(I)

CALL COPY(X(2),2,Y(9),3,10)
This is equivalent to:      DO 5 I=1,10
                          5 Y(3*I+6)=X(2*I)
```

Subroutine: *CALL INIT(Z, NTOTAL, ZVALUE)*

- Z — Array element or array name
- NTOTAL — Total number of Z-element to be initialized
- ZVALUE — Common value

If Z is an array name, initialization begins from Z(1). If Z is an array element, initialization begins from the given element. Z must be a single precision, real or integer type.

Effect: Initialize the array by the common value given

Example:

```
CALL INIT(X,50, 0.0)
Set  X(1),X(2),...X(50) to zero.

CALL INIT(K(10), 50, KODE)
Set K(10),K(11),...K(59) to a pre-defined value KODE.

CALL INIT( POINT, 132, ' ')
Set POINT(1),POINT(2),...POINT(132) as blanks.
```

Function or Subroutine:

MINX (ITEM(I) , ITEM(J) , INDEX)
MAXX (ITEM(I) , ITEM(J) , INDEX)
AMINX (ITEM(I) , ITEM(J) , INDEX)
AMAXX (REAL(I) , REAL(J) , INDEX)

First element in the specified array, integer or real as indicated.

Last element in the specified array, integer or real as indicated.

Order of Min or Max element in the specified list.

Effect: As a subroutine, it returns as *INDEX* the order of the minmax number in the given array. The actual subscript of the minmax element and the value of that minmax will require additional computation:

subscript of the minmax element = I + INDEX -1
MINMAX = ITEM(I+INDEX-1) or REAL(I+INDEX-1)

As a function, it only returns the value of the minmax element. The subprogram is not applicable to double precision or complex list.

Example: *CALL AMAXX(X(3),X(300),INDEX)*
If the subroutine returns a value of INDEX as 59, then the maximum of the X-list is X(61).

(3) Subprogram to control TTY characteristics

This subprogram will accomplish at execution-time a control of terminal characteristics properties in the same manner of what the monitor command "SET TTY" can accomplish at the monitor level. In a monitor command "SET TTY" (or "TTY" in its short form), the general form is: TTY *keyword* , where *keyword* is either one of a complementary pair of arguments, such as *PAGE* or *NO PAGE*. In the subprogram shown here named as SETTTY, the "PAGE" part of the example is called a Code Parameter, and yes-or-no part is called a Logic Parameter. Thus the entire group of TTY commands can be coded into a single subroutine. This is shown next.

Function or Subroutine: *SIXBIT (Z, I, J)*

- J — number of character to be converted to the SIXBIT code
- I — Destination of character after conversion
- Z — Source of ASCII character to be converted

Effect: When used as a subroutine, it returns an array I which is the SIXBIT code of Z. If it is used as a function, the first 6 characters (padded with blanks if necessary) is returned as the value of the function.

Note: Both Z and I are dimensioned variables for the same ASCII characters. However, SIXBIT codes contain six characters per word, while the ASCII codes contain five characters per word. So, the dimensions of Z and I could be different.

Example: *CALL SIXBIT('SYS', IDUM, 3)*
Convert the ASCII string 'SYS' into SIXBIT code as IDUM.

Subroutine: *CALL RUN(DEVICE,SAVEFILE,PPN)*

- PPN — PPN (octal) where file is stored. PPN=0 if in own disk.
- SAVEFILE — Octal number, SIXBIT code of filename of the EXE file to be run.
- DEVICE — SIXBIT code of the device (no colon)

e.g. DSK = "446353000000
(or = "0)
DTA0 = "446441200000
SYS = "637163000000

Effect: This is equivalent to STOP for the current program; then apply a monitor command of ".RUN DEV:NAME[m,n]".

If DEVICE='SYS', 'NEW' or 'OLD' in SIXBIT codes, then PPN=0. If DEVICE='MT7', 'MT8', or 'MT9' in SIXBIT code, the tape must be already properly mounted and positioned.
The RUN subroutine will drop all files in the old program. If files in the old program are dropped without first a CALL RELEAS call, the files will be lost if they are output files, and will not be available as intermediate data for running the chained programs.

Example: *CALL RUN(0,SIXBIT('DEPT',IDUM,4),"115103320571)*
This is equivalent to STOP the current program and then issue a monitor command of "RUN DEPT[115103,320571]"

For the convenience of users and by the permission of Mr. Nicholas, a copy of the SUBSET package is stored also in ENG: , which is the depository of the Engineering Program Library.

Since SUBSET is not in the FORTRAN-10 Library but in the user-library the EXECUTE command of a FORTRAN program should specifically includes "ENG:SUBSET.REL/LIB" in its list, if the program calls any subprogram in the SUBSET package. In a batch job, a $INCLUDE card is necessary. For example, the following is an execution command for a program that calls the SUBSET subprograms:

```
.EXECUTE MAIN.FOR, SUB1.FOR, ENG:SUBSET.REL/LIB
```

3.23 Comprehensive FORTRAN Subroutine Libraries

In an academic user community of the size of the University of Pittsburgh, it has been estimated that more than 500 "new" Gaussian Elimination programs for simultaneous equations were written, debugged, and run each year. Many of these came out of courses in programming, numerical methods, engineering analysis, economics, statistics, etc. Most of them are justifiable as they provide the students opportunities to sharpen their skill on a familiar problem with proven methods of solution. But some were unnecessary exercises to "re-invent the wheels" since the elements of student learning are absent in those exercises. Such activities are pure waste of human resources and computer resources.

It may be said that computer applications in radically different disciplines share a common ground that an application must be first mathematically formulated. Once so done, the differences between disciplines disappear. For example, the Gaussian Elimination method would be applicable whether the problem was orginated from a power system load flow study or a regression study from an economics model, so long as the problem is formulated as a system of linear simultaneous algebraic equations. Thus a software package containing standard solutions to various mathematical problems is a very useful tool to computer users in all disciplines.

In order for such a software package to serve a large group of users in many diversified fields, there are several important requirements that must be satisfied:

(1) These programs should be callable in the forms of subprograms (subroutines or functions), so that the user's program remains in control.

(2) These subprograms should be self-contained so that they will not require further attention from the users other than passing the values of the subprogram parameters into the subprograms. In particular, there should not be any input/output statements in the subprogram. Thus the input/output operations become the responsibility of the user's main program. There are exceptions, of course. A subprogram may be designed explicitly for input or output operations, for example, to list and tabulate a matrix.

(3) In order to adapt to the need of different users, each subprogram should have capability of adjustable dimension size as well as user-controllable error level. At least an estimate of error level should be available as a return value of the subprogram, so that the user, who has no knowledge of how this subprogram was constructed, will know the level of performance of the program.

(4) There should be clear and uniform documentations available to guide the users in defining the subprograms, including the dummy parameters, their types, array sizes, order in the parameter list, and their meaning.

At the University of Pittsburgh, two such packages are available. One is the International Mathematical & Statistical Library (IMSL) which in on-line as PRG:IMSL.REL. The other is the IBM Scientific Subroutine Package (SSP)*, which is not on-line but may be placed on-line by running a UARC program, as it to a great extent duplicates the IMSL coverage. Both packages are comprehensive in their coverage, and their documentations are excellent but voluminous. However, when a user is faced with a big programming job whose purpose may be mode than a programming exercise, it will be cost-effective to use these library facilities, even to the extent of modifying the program in order to fit.

Both IMSL and SPP contain several hundred subprograms in the package, and therefore are too voluminous to include in this bbok even in a summarized form. Only the areas of coverage will be given here to give the readers some idea about the comprehensiveness of the package:

IBM SSP Package:

Statistics:

Probit analysis
Variance analysis
Correlation analysis
Multiple linear regression
Polynomial regression
Canonical correlation
Factor anaysis
Discriminant analysis
Time series analysis
Data screening and analysis
Nonparametric tests
Random number generation
Distribution functions

Mathematics:

Inversion
Eigenvalues and eigenvectors
Simulataneous linear algebraic equations
Transpositions
Matrix arithmetics
Matrix partitioning
Matrix tabulation and sorting of rows or columns
Elementary operations on rows or columns of matrices
Matrix factorization
Integration and differentiation of given or tabulated functions
Solution of systems of first-order differential equations
Fourier analysis of given or tabulated functions
Bessel and modified Bessel function evaluation
Gamma function evaluation
Jacobina elliptic functions
Elliptic, exponential, sine cosine, Fresnel integrals
Real roots of a given equation
Real and complex roots of a real polynomial equation.
Polynomial arithmetic
Polynomial evaluation, integration, differentiation

*For Pitt users, the SSP source programs are stored and available on a UARC tape B4473. See Section 10.7 for the UARC procedure.

Chebyshev, Hermite, Laguerre, Legendre polynomials
Minimum of a function
Approximation, interpolation, and table construction

IMSL Package: Chapter headings:

Analysis of Variance
Basic Statistics
Catagorized Data Analysis
Differential Equations; Quadrature; Differentiation
Eigensystem Analysis
Forecasting; Econometrics; Time Series; Transforms
Generation and Testing of Random Numbers
Interpolation; Approximation; Smoothing
Linear Algebraic Equations
Mathematical and Statistical Special Functions
Non-Parametric Statistics
Observation Structure; Multivariate Statistics
Regression Analysis
Sampling
Utility Functions
Vector, Matrix Arithmetic
Zeros and Extrema, Linear Programming

Example: Suppose we are to solve a system of 50 simultaneous equations. In matrix form, the equation is Ax=B. Suppose the matrices have been stored as DATA.DAT file with a format of (10E12.4). In the file, the first 250 records are the A-matrix by rows, and the last 5 records are the B-matrix. Obtain the solution by using the IMSL package.

Program: The first step of this problem is naturally to search through the IMSL documentation to see if there is one that fits the problem. Such a problem would of course be under the category of "Linear Algebraic Equations." When such a program is found, the user's task is to prepare a main program which calls this IMSL routine. To do so, the main program will include the following parts:

(1) To provide storage (the DIMENSION statement) for all variables required for the problem. This not only includes the problem variables but also the working variables. The IMSL documentation gives detailed and exact requirements of DIMENSION.

(2) To input the data needed by the Library subprogram. This includes opening of files, reading of data from file or terminal, calculations needed for the subprogram parameters, etc.

(3) To call the IMSL subprogram.

(4) To output the results.

IMSL Reference Manual (Reference 10) is a seven-inch thick reference book. The content is divided into 17 chapters, and Chapter L is on Linear Algebraic Equations. In going through the routines in that chapter, the routine

LEQT1F lists the following headings:

```
IMSL ROUTINE NAME - LEQT1F
PURPOSE           - LINEAR EQUATION SOLUTION - FULL STORAGE
                    MODE - SPACE ECONOMIZER SOLUTION
```

This seems to satisfy our need. The other information listed by the Manual are included below:

USAGE - CALL LEQT1F(A,M,N,IA,B,IDGT,WKAREA,IER)

A - Input matrix of dimension N by N containing the coefficient matrix of the equation Ax=B. On output, "A" is replaced by the LU decomposition of a rowwise permutation of "A".

M - Number of right-hand matrix columns (input)

N - Order of "A" and number of rows in "B".

IA - Row dimension of A and B exactly as specified in the DIMENSION statement of the calling program.

B - Input matrix of dimension NxM containing right-hand side of the equation Ax=B. On output, the NxM solution X replaces B.

IDGT - Input option: If IDGT>0, the elements of A and B are assumed to be correct to IDGT decimal digits and the routine performs an accuracy test. If IDGT equals zero, the accuracy test is bypassed.

WKAREA - Work area of dimension >= N.

IER - Error parameter (output).
Terminal error: IER=129 indicates that matrix A is algorithmically singular.
Warning error: IER=34 indicates that the accuracy test failed. The computed solution may be in error by more than can be accounted for by the uncertainty of the data. This warning can be produced only if IDGT is greater than 0.

In checking over these specifications, the following should be noted:

(1) The matrices A and B will be destroyed after the execution of the subprogram. If they are needed later, protect them by copying them into another set of variables, or else later re-read the input data A and B.

(2) The DIMENSION for the storage declaration should be A(IA,IA), B(IA,M). In addition, it is also the responsibility of the calling program to dimension WKAREA(IA). Note that B is dimensioned as a matrix with two subscripts. If B is a vector, as in most linear systems, B should be dimensioned as B(IA,1).

(3) N and IA need not be the same, but N should never exceed IA. If N is an input quantity and made to be less than IA, such a calling program would be able to solve a system of linear algebraic equations of an order specified by the user up to IAth order. Such a program would increase its flexibility immensely.

The program for this problem is listed below:

```
      DIMENSION A(100,100),B(100,1),WKAREA(100)
***** DEFINE THE SIZE OF PROBLEM "N"
      WRITE(6,100); READ(5,101)N
100   FORMAT(/' ENTER NUMBER OF VARIABLES = '$)
101   FORMAT(I)
***** GET INPUT DATA FOR THE SUBPROGRAM
      OPEN(UNIT=1,FILE='DATA.DAT',ACCESS='SEQIN')
102   FORMAT(10E)
      DO 10 I=1,N
10    READ(1,102) (A(I,J),J=1,N)
      READ(1,102) (B(I,1),I=1,N)
***** CALL IMSL SUBPROGRAM LEQT1F
      M=1; IA=100; IDGT=0  !SUBROUTINE PARAMETERS
      CALL LEQT1F(A,M,N,IA,B,IDGT,WKAREA,IER)
***** OUTPUT THE RESULTS
103   FORMAT(/' X(',I2,') = ', E12.4)
      WRITE(6,103) ((I,B(I,1)), I=1,N)
      STOP
      END
```

Suppose we name the stored program EQUAT.FOR. This program may be executed by a monitor command of:

```
.EXECUTE EQUAT.FOR, PRG:IMSL/LIB
```

With the dimension set up in EQUAT.FOR, it is capable to solve a system of up to 100 equations. However, when solving a large system, the accuracy requirement may be difficult to satisfy because of the accumulation of round-off and truncation errors during computations. Then the accuracy test would fail in the subroutine execution, giving the output IER a non-zero report.

The following is the computer printout of the execution:

```
.EXECUTE EQUAT.FOR, PRG:IMSL/LIB
FORTRAN 5A(621): EQUAT.FOR
MAIN.   OCTAL PROG SIZE=24167
LINK:   Loading
[LNKXCT EQUAT execution]

ENTER NUMBER OF VARIABLES = >100

X( 1) =  0.9996E+00

X( 2) =  0.1996E+01

X( 3) =  0.3000E+00

X( 4) =  0.4000E+00

  etc.....
```

3.24 Array Processor

In many engineering and scientific applications, the computations often involve a relatively simple algorithm done repeatedly on log sequences of data. The data may be one-dimensional sequence of numbers (called vectors), or two or more dimensional sequences, (called arrays), for example, a matrix. In such computations, heavy overhead must be absorbed on such "book-keeping" chores of array indexing, loop counting, and data fetching. In conventional computer organization, such overhead must be absorbed by incorporating them sequentially into the program, thus competing for machine time with the actual computations.

The concept of parallel processing is to provide hardware so that independent computations can be performed at the same time and result in a much faster program execution.

At the DEC-10 installation at the University of Pittsburgh, one such parallel processor, the Floating Point 190L Array Processor, is attached to the System B. The AP190L is a pipe-line machine that allows the calculations of overhead for elements up stream to be performed simultaneously with the element computations down stream (therefore, the name pipe-line).

To the FORTRAN users, the usage of the AP190L means to incorporate certain AP190L subroutines calls in the main program. Thus, writing a FORTRAN program that uses the array processor to process data follows the general rules of FORTRAN subroutine calls. There are a few exceptions:

(1) The array processor must be initialized before using other AP190L subroutines.

(2) Data must be transferred from DEC-10 to AP190L main data memory before the array processor can operate on it.

(3) In order to synchronize the AP190L with the DEC-10, wait calls (in FORTRAN subroutine) must be inserted in the program whenver the DEC-10 and AP190L interact.

(4) At the end of array processor execution, data must be transferred back to DEC-10.

All of these steps are done by calling certain appropriate AP190L subroutines. These subroutines are listed and explained in details in Reference 11. The AP190L Math Library contains subroutines distributed in the following areas:

(1) Data transfer and control operations

(2) basic vector arithmetic

(3) Vector-to-scalar operations

(4) Vector comparison operations

(5) Complex vector arithmetic

(6) Data formatting operations

(7) Matrix operations

(8) Fast Fourier Transform operations

(9) Auxiliary operations

(10) Utility operations

(11) signal processing operations

(12) Table memory operations

Users should consult Reference 12 concerning the usage of the AP190L. Specifically, note the following:

(1) AP190L is attached to DEC-10 System B as a periperal device. Therefore, just as a tape unit, it requires the "DRIVE" monitor command to reserve it. See Section 8.10.

(2) It requires large core memory, larger than most time-sharing allocations. Therefore, array processor runs should be submitted as batch jobs. See Chapter 9 on how to submit batch jobs.

3.25 FORTRAN 77

The FORTRAN programming language is one language that is universally available, on computers, large or small, in the United States, Europe or the rest of the world. Thus, its greatest contribution is that a program written in FORTRAN can be run on any machine, after some minor modifications are made if required.

The ANSI FORTRAN IV, standardized by ANSI in 1966, has exercised a powerful influence on the portability characteristics of the language. In the past fifteen years, there have been many enhancements of the ANSI standard, and FORTRAN-10 is one such enhancement. Varieties of these enhanced versions generate a new need for standardization. Thus, an updated standard language was announced in 1977, unofficially known as FORTRAN 77, and was formally standardized in 1978 by ANSI (ANSI Standard X3.9-1978). While compliance with the ANSI standard is voluntary, it is expected that all FORTRAN languages will be in time evolved into this new version. By necessity, programming languages must have universal portability, and the ANSI standard has powerful influences. FORTRAN-10 already possesses most of the new attributes of FORTRAN 77, but many keywords and syntax are different. It is expected that in a few years, FORTRAN-10 will be replaced by some version of FORTRAN 77. Details of FORTRAN 77 are outside the scope of this book. Interested readers are referred to References 13 and 14 for more details.

REFERENCES

1. PROGRAMMING WITH FORTRAN, Byron S. Gottfried, Quantum Publishers, New York; 1972.

2. PROBLEM SOLVING AND STRUCTURED PROGRAMMING IN FORTRAN, F. L. Friedman and E. B. Koffman, Addison-Wesley Publishing, Reading, Massachusetts; 1977.

3. DEC SYSTEM-10 FORTRAN-10 LANGUAGE MANUAL, Second Edition, DEC-10-1FORA-B-D, Digital Equipemnt Corporation, Maynard, Massachusetts; 1974.

4. DEC SYSTEN-10 FORTRAN PROGRAMMER'S REFERENCE MANUAL, AA-0944E-TB, Digital Equipemnt corporation, Maynard, Massachusetts; 1977.

5. FORTRAN-10 USERS GUIDE, Computer Center, University of Pittsburgh, Pittsburgh, Pennsylvania; 1977.

6. PITT Programmer Notes, Special FORTRAN-10 Issue, Vol. 6, No. 5, August 1, 1977, Computer Center, University of Pittsburgh, Pittsburgh, Pennsylvania; 1977.

7. SUBSET MANUAL, Ronal K. Nicholas, University of Pittsburgh, Pittsburgh, Pennsylvania; 1977.

8. SYSTEM/360 SCIENTIFIC SUBROUTINE PACKAGE (360A-CM-03X) PROGRAMMER MANUAL, IBM Corporation, White Plains, New York.

9. Help File PRG:IMSL.HLP, the Computer Center, University of Pittsburgh, Pittsburgh, Pennsylvania; 1980.

10. IMSL LIBRARY REFERENCE MANUAL, Edition 7, International Mathematical and Statistical Library, Houston, Texas; 1979.

11. AP MATH LIBRARY MANUAL, Volumes 1,2,3, Floating Point System, Inc.; 1979.

12. Help File PRG:APU.HLP, Computer Center, University of Pittsburgh, Pittsburgh, Pennsylvania; 1980.

13. FORTRAN 77, FEATURING STRUCTURED PROGRAMMING, L. P. Meissner and E. I. Organick, Addison-Wesley Publishing Company, Reading, Massachusetts; 1980.

14. PROGRAMMING IN STANDARD FORTRAN 77, A. Balfour and D. H. Marwick, North-Holland Inc., New York, New York; 1979.

CHAPTER 4

FORTRAN PROGRAM DEBUGGING

4.1 Introduction

One of the most important but unpleasant stage in the computer usage is the necessity to debug a program. The development of programs and the subsequent computer execution involve a long chain of events that requires error-prone human actions. These errors can be committed by beginners as well as by experienced users. The detection and the correction of such errors affect seriously the productivity of computer processing applications. These errors are colloquially referred to as "bugs", and the process of detecting and correcting them as "debugging."

The following are some typical statistics regarding the productivity of professionals in the software industry:

The average productivity of a professional programmer in U.S. is seven (7) FORTRAN statements per working day.

For the software development done at a commercial software firm, 65% of the software cost is attributed to debugging.

Breakdown of computer processing failures: (From Reference 1)

Hardware failure	1%
System software failure	2%
Operator mistakes	5%
System failure	2%
Programming errors	90%

It becomes increasingly obvious in the commercial software industry that debugging is by far the major component of the software cost. Conversely, when a software is developed on a fixed budget, the extent of testing and debugging becomes the deciding factor for the software product reliability. In the recent decade, considerable efforts have been spent on the optimal allocation of resources, design of software structure for easy testability and maintainability, test and validation procedures for softwares, and various diagnostic aids, resulting collectively in a new discipline known as "software engineering."

Unfortunately, in spite of advances in the software engineering practices, the degugging of a computer program still depends heavily on the user's knowledge and experiences in the problem, the language, and the computer, and hence it still remains largely as an art. However, over the years, accumulation of expertise and experience has resulted in the formulation of reliable guide lines, good programming styles and practices, checklists for DO's and DON'T's, error reporting and diagnostic facilities in the language processors, and on-line debugging tools. It is, therefore, the purpose of this chapter to present a summary of these practices, with particular emphasis on FORTRAN program debugging.

4.2 Types of Errors

When a FORTRAN program fails, a very natural inclination of the user is to suspect that "the computer is acting up again." Mercifully, the computer system hardware and system software failures are quite rare nowadays, and program errors can usually be blamed as the culprits.

Program errors are the most numerous and also the most complicated. They may be divided into the following categories:

(1) Errors in problem definition They are errors resulted from failures to translate the problem requirement faithfully into the program requirements.

(2) Coding errors They appear in several different forms:

a. Transcription errors, such as incorrect punctuations and misspellings. Such errors will usually be caught at compiling, but some errors may go undetected as perfectly legal program statements and a compiler may not always be able to spot them.

b. Syntax errors, or improper use of FORTRAN statements. Such errors can usually be detected by the compiler.

c. Structural errors or failures to provide correct interaction between two parts of a program, for example, failure to pass the values of parameters from the main program to a subprogram correctly.

(3) Logic errors. These are failures to sequence the problem properly at a detailed level.

For the remainder of the chapter, we will be mainly concerned in two areas of the debugging process:

(1) How can we reduce the incidence of all types of bugs?

(2) If a bug exists, how do we detect and correct it?

PRE-COMPUTER-RUN DEBUGGING

The most effective way of minimizing program errors is not making them in the first place. Since debugging constitutes 60-90% of a user's effort in computer processing, it is cost effective to spend extra effort and time in keeping good practice of preparing programs so that the debugging time will be reduced.

At this stage, we will assume that the user understands his problem thoroughly and translates it faithfully into the program requirements and objectives. If a user fails to do that, no amount of debugging effort can rectify the situation. Thus, we will focus our attention to the bugs that are either the coding errors or the logic errors or both.

4.3 Walkthrough by Flow Charts

One of the most neglected good practice is the preparation of a flow chart, before any coding is done, to lay out the flow logic of the problem. After the problem is coded, the program will be burdened with a jumble of statement details, and the problem logic is then obscured. A flow chart is a graphical representation of the logic flow, and is a valuable tool as a problem record as well as a tool to identify potential errors.

Basic mechanics of flow charting is a part of introductory training of computer programming and will not be repeated here. For more details, the readers are referred to any standard FORTRAN manual or References 4 and 5.

"Walkthrough" check of a problem is to check the flow logic by playing the computer in tracing out the steps of computer processing. Playing computer in tracing step-by-step in the program is a tedious chore, but the job may be made easier by tracing the steps on a flow chart.

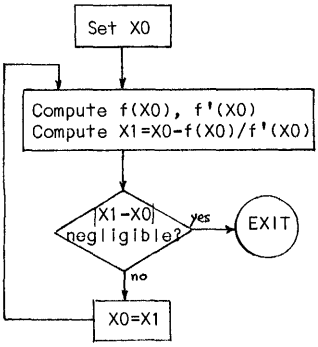


Figure 4.1

Let us consider the case of designing a subprogram for the Newton-Raphson's method of solving for a real root of a polynomial equation with no multiple roots.

Algorithm: Given a polynomial equation of order n:

$$f(x) = A_n x^n + A_{n-1} x^{n-1} + \ldots + A_1 x + A_0 = 0$$

The iterative formula is:

$$X1 = X0 - f(X0)/f'(X0)$$

To start the iteration, set X0 = certain trial value. At the end of X1 calculation, X1 and X0 are compared. If they differ negligibly, return the result and exit. If they differ substantially, use X1 as the new X0, and the X1 is recalculated. And the whole process is repeated.

This wordy description may be greatly clarified by a flow chart, as shown in Figure 4.1.

Now we are ready to do a flow-chart walkthrough.

First, we check if all the flow arrows go to the right places. Playing the computer, we assume a set of data, then follow and trace the flow of computations.

If there is any discrepancy, corrections can be made right there before coding the program. If everything seems all right, the next step is the most important one. We ask: Under what circumstances can the flow chart go wrong?

There are many such circumstances. For example:

(1) Even-order equation may not have any real root. Under that circumstance, this method shows that the computer will keep on iterating without end, causing an endless loop trying to find a non-existent real root. Has there been any provision in the flow chart for such contingency? Answer: No!

(2) If X0 is set arbitrarily, what if it has a value that will make f'(X0)=0? That will create a division-by-zero situation. Has the flow chart indicated how that can be detected and handled? Answer: No!

(3) Even when the iteration does converge, the flow chart does not show any control over its efficiency. For example, if it takes more than 1000 iterations to converge to a solution, do we want this method, or should we try another method?

We have identified only a few weak spots. There are more when we analyze further. For example, this subprogram requires pre-setting a trial value for X0. Why not generate it automatically inside the subprogram? But how? One way is to compute -A(n-1)/A(n) and call the result first-trial X0. Then what happens when A(n)=0, or can it happen? Thus, walking-through the flow chart playing a devil's advocate, we can identify and strengthen the weak spots of the logic and substantially improve the reliability of the program generated from it.

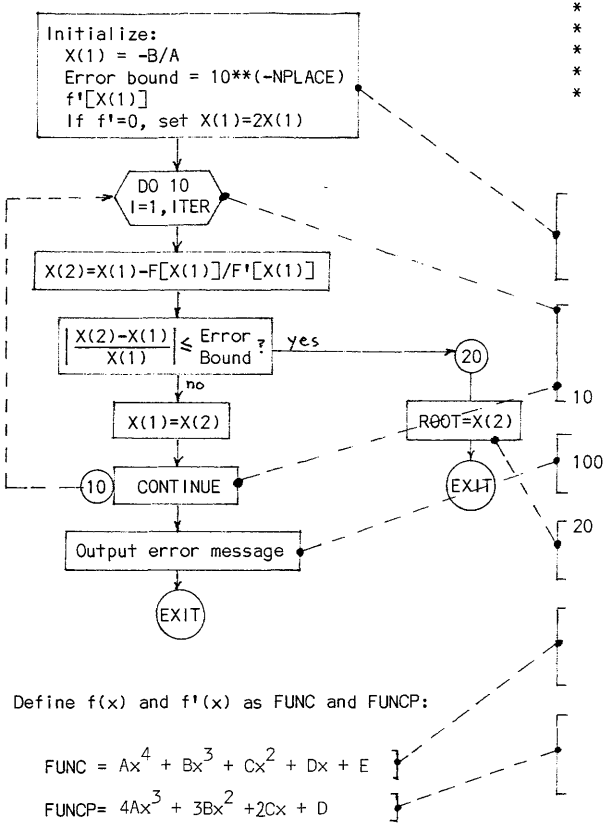
Figure 4.2 shows the revised flow chart after the walk-through process. It is not perfect yet, but its reliability is much more improved than the first one. Block by block, the flow chart blocks are coded into a FORTRAN program, and this is shown along side with the flow chart. To keep it simple at this stage, we use a 4th order polynomial equation as an example.

There are many other benefits of using flow charts:

(1) A flow chart can identify more easily the inter-relations between parts of a program, and can therefore be used effectively in partitioning the program into modules. Later, the testing and the debugging can be made more effective by modularizing the complete program.

(2) A flow chart can serve as a language-independent record of the program logic. Should there be any modification or adaptation of the program using a different language, the flow chart can serve as a generic specification of a computer program.

(3) A flow chart can be re-generated from a finished or existing program. Thus, if we compare the flow chart specification of the problem with the flow chart derived from the resulted program, discrepancy and logic error can be quickly discovered and corrected without costly computer runs.




```
*     To find a real root "ROOT" to an accuracy of "NPLACE"
*       decimal places within "ITER" iterations, using the
*       Newton-Raphson's method for the equation:
*
*       Ax**4 + Bx**3 + Cx**2 + Dx + E = 0

      SUBROUTINE NEWTON(A,B,C,D,E,ITER,NPLACE,ROOT)
      DIMENSION X(2)

      X(1)=-B/A
      ERROR=10.**(-NPLACE)
      F1=FUNCP(A,B,C,D,X(1))
      IF(F1.EQ.0.)X(1)=2.*X(1)

      DO 10 I=1,ITER
      X(2)=X(1)-FUNC(A,B,C,D,E,X(1))/FUNCP(A,B,C,D,X(1))
      IF(ABS((X(2)-X(1))/X(1)).LE.ERROR)GOTO 20
      X(1)=X(2)
   10 CONTINUE

      WRITE(6,100)ITER
  100 FORMAT(/'*** NO CONVERGENCE WITHIN',I5,' ITERATIONS.')
      RETURN

   20 ROOT=X(2)
      RETURN
      END

      FUNCTION FUNC(A,B,C,D,E,X)
      FUNC=A*X**4+B*X**3+C*X**2+D*X+E
      RETURN
      END

      FUNCTION FUNCP(A,B,C,D,X)
      FUNCP=4.*A*X**3+3.*B*X**2+2.*C*X+D
      RETURN
      END
```

Figure 4.2 Correspondence Between a Flow Chart and a Program

(4) Even if the original flow chart is not available, as is the case with most beginners, generating a flow chart from the finished program will aid in inspecting the flow of program logic. Even though a program may be well documented and commented, it may be too long and complicated to discover the errors easily by just reading the program.

Generating a flow chart from an existing program manually is also a tedious task. Fortunately, most computing facilities have a software facility to generate a flow chart by computer. Such a software is available on the DEC-10 at Pitt. The program description for FORFLO is given in the next section.

4.4 The FORFLO Program

The FORFLO program is a program stored in the device "PRG:" (the System Program Library), and it is used to process a finished FORTRAN source file. It has the following capabilities:

(1) To relabel the statements in a FORTRAN source file. Statements are given new numbers as their labels in ascending sequence, and the format statements are moved and assembled at the end of the program.

(2) To create a flow chart of the source program on a listing file.

(3) To re-format the source program into 80-column records.

(4) to resequence columns 73-80 of each card.

Since the FORFLO is a System Program Library program, the monitor command need to run it is "RUN". Note that programs such as PIP, UPDATE etc are the System programs, and use the monitor command "R" to run them. Thus, the command to call the FORFLO program is:

```
.RUN  PRG:FORFLO
```

The System will respond with a message "FORFLO V.05 /H FOR HELP." and a prompting symbol of "*". At this point, the user may either type "/H" to get the HELP-file (5-page long, about 15 minutes typing time), or apply a FORFLO command.

The standard format of a FORFLO command is:

```
REVISED-FILE, LISTING-FILE = SOURCE-FILE/Optional Switches
```

where REVISED-FILE = output file where the revised source program will be stored. It may be omitted if not needed.

, = a comma, required to separate the revised and the listing files.

LISTING-FILE = name of the listing file.

SOURCE-FILE = name of the FORTRAN source file to be processed.

Each file has the standard specification of DEV:NAME.EXT. The PPN designation [m,n] is allowed in the file specification of the SOURCE-FILE. Standard default conditions also apply. Default extension of the REVISED-FILE is FOR, and that of LISTING-FILE is LST.

Either or both of the REVISED-FILE and the LISTING-FILE may be omitted in the command. The meaning of omission is as follows:

(1) If the REVISED-FILE part of the command is omitted, the command becomes:

LISTING-FILE = SOURCE-FILE/Switches

and the revised file will not be made. Note the required leading comma.

(2) If the LISTING-FILE part of the command is omitted, the command becomes:

REVISED-FILE, = SOURCE-FILE/Switches

and the listing file will not be created. Note the required comma. Since the cross reference table and the flow chart are available only on a listing file, this command will only revise and store the source program, but it will not produce a flow chart or a cross reference table.

(3) If both the REVISED-FILE and the LISTING-FILE are omitted, the FORFLO will check the source file for those errors that FORFLO is able to detect, and report the errors on the user's terminal. All other functions are omitted.

Options and variation from the standard format can be specified by including optional switches. The list of switches and their functions are given in Table 4.1.

Example: The example used in Section 4.3 will be used here for illustration. The program is stored on disk as NEWTON.FOR. The following shows the sequence of calling and executing the FORFLO program:

```
.RUN PRG:FORFLO

FORFLO V.05 /H FOR HELP.
*NEW.FOR, NEW.LST = NEWTON.FOR/Y/100F
FORFLO: NEWTON
*^C
```

The listing file is produced at the printer and then is deleted from the user's storage quota by a command:

```
.QUEUE NEWTON.LST/DISPOSE:RENAME
```

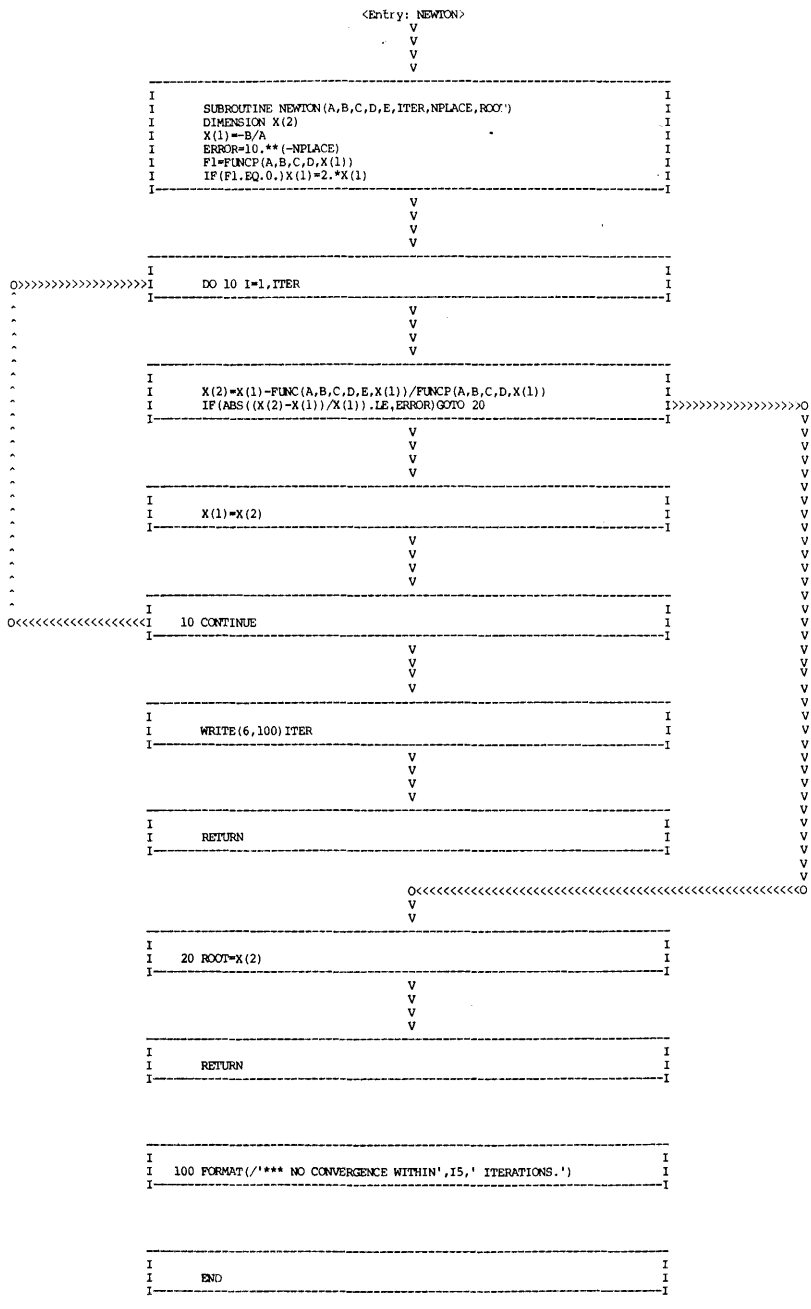
If the user wishes to reproduce the list file on his terminal he can do so by a command:

```
.TYPE NEWTON.LST/EMULATE:LABELS.CCT
```

Option Switch	Function	Default Condition
/nnnF	Format reshuffle with user-specified format number starting from nnn.	FORFLO assigns format number.
/nnnL	Sequencing of statement labels will be consecutive multiples of nnn.	nnn=10; /L and /0L are both treated as /10L.
/nnnS	Put sequence numbers on the REVISED-FILE in increments of nnn. These numbers will be in columns 73-80.	nnn=0; no number is placed in col. 73-80.
/A	To make a flow chart (implying /C); LISTING-FILE must be specified.	
/B	To delete blank lines in the SOURCE-FILE.	Keep the blank lines.
/C	To make a cross reference table between the source program and the revised program.	
/H	To type out the HELP file.	
/K	To change 026 punches to 029 punches.	
/Q	To suppress all listing except the flow chart.	
/T	No tab conversion to blanks.	Tab in the source file changed to appropriate number of spaces.
/W	To suppress warning message when line truncation in the revised program is encountered.	
/X	Flow chart only; equivalent to the compounded switches of /L/A/S/T/Q. No revised program; just the flow chart.	
/Y	Do everything: move formats to the end of the program, resequence the statement numbers, and make a flow chart.	

Table 4.1 FORFLO Switches

where "LABELS.CCT" should be a file with a content of "/DC3:1-66:1" in it. This will allow your terminal to emulate a printer and interpret certain printer-carriage control characters. Since terminal typing is slow and the list file is usually long, this practice should be used sparingly, for example, only when a printer is not available to you. The flow chart for NEWTON.DAT in the example is reproduced in Figure 4.3. This should be compared with the flow chart in Figure 4.2 to see if there is any discrepancy.


<Entry: NEWTON>
SUBROUTINE NEWTON(A,B,C,D,E,ITER,NPLACE,ROOT)
DIMENSION X(2)
X(1)=-B/A
ERROR=10.**(-NPLACE)
F1=FUNCP(A,B,C,D,X(1))
IF(F1.EQ.0.)X(1)=2.*X(1)
DO 10 I=1,ITER
X(2)=X(1)-FUNC(A,B,C,D,E,X(1))/FUNCP(A,B,C,D,X(1))
IF(ABS((X(2)-X(1))/X(1)).LE.ERROR)GOTO 20
X(1)=X(2)
10 CONTINUE
WRITE(6,100)ITER
RETURN
20 ROOT=X(2)
RETURN
100 FORMAT(/'*** NO CONVERGENCE WITHIN',I5,' ITERATIONS.')
END

OFF-LINE DEBUGGING BY CODE INSPECTION

On-line debugging is an expensive process as it uses substantial amount of computer resources. As a result, it should be reserved as the last resort in the sequence of debugging process.

One effective off-line debugging process adopted at many software houses is debugging by team inspection of the code. A typical team consists of four people, with one person, not the author of the code, acting as the moderator. At the inspection time, the programmer is to narrate the code, statement by statement, the logic of the program. The other team members not necessarily are familiar with the problem, but they have opportunities and time to study the code. At the inspection period, questions are raised and potential error sources are pursued. The organization would normally have an error checklist for common programming errors, and that list is thoroughly gone over.

At the conclusion of such a team inspection period, a list of errors and suggestions of how to correct them is drawn up. The programmer is then to carry out the correcrions. The value of team inspection is to inspect the program with a fresh point of view to avoid the "forest-tree syndrome." ("When you are in the forest, you don't see the trees.") In an academic environment, a group of 3-4 students can form an inspection team. This is not only an effective procedure, but also a valuable learning experience from each other.

For the team inspection practice, a valuable aid is a checklist for historically common programming errors of the particular programming language. For the remainder of this section, we will go over such lists that pertain to FORTRAN-10.

The common programming errors can be categorized into the following types:

(1) Data errors.

(2) Computation errors.

(3) Logic errors.

(4) Input/Output errors.

The checklists for these types of errors will be outlined next. In these checklists, we will not deal with any transcription errors, such as incorrect punctuations, wrong spelling, illegal characters, or anything that violates the rules of FORTRAN-10 language. We assume that proof-reading of the codes has been done thoroughly by the programmer at this stage, and all "typos" have been corrected.

4.5 A Checklist for Data Errors

This type of errors pertains to data incorrectly referenced, declared, typed, initialized, or set. Itemized below are a number of commonly committed errors presented in the form of a checklist.

(1) Does any of the array subscript exceed the bounds specified in the DIMENSION statement?

Among common programming errors, out-of-bound subscript is probably one of two most frequently committed errors, by both beginners and experienced users. If this type of error is not caught at the code inspection time, it may be very difficult and costly to detect it later. A principal reason is that an out-of-bound error will cause an unpredictable error symptoms at execution time. These symptoms are actually secondary effects of the error. What makes it difficult to diagnose during the on-line debugging is that a secondary symptom may not give any hint of an out-of-bound error. Observe the following example:

Example: One segment of a FORTRAN-10 program is as follows:

```
      DIMENSION K(100)
      DO 10 I=1,110     !***Subscript out of bound error here
10    K(I)=I
      TYPE 50, K(5)
50    FORMAT(/' K(5)= ', I3)
      ...
```

The error message from the program execution reported that there is an illegal character "^@" in the FORMAT statement. This is actually caused by the invasion of K-values into the FORMAT area in the execution program. Thus, in a more complicated program, one can waste a great deal of time and resources trying to track down the "FORMAT" error, if the error message is taken at its face value blindly.

(2) Is a variable referenced in a statement previously initialized, set, and with its type implicitly or explicitly declared?

This is the other one of two most frequently committed programming errors. FORTRAN-10 automatically set a variable storage area to zero, unless there are DATA statements to set them to other values. Thus, many such errors go undetected and cause no damage. However, if that segment of program is used more than once, there will be initialization errors from the second round on, because the value of the variable, if not initialized, will be its result value of its last calculation. Thus the variable is inadvertently initialized to a wrong value. For FORTRAN-10, an incorrectly initialized variable can cause chaos if that happens to be, for example, computed GO TO index or a variable used in an IF statement. The following shows an example of calculating the sum of five integers.

```
      DIMENSION K(5)
1     READ(5,5)(K(I),I=1,5)
5     FORMAT(5I)
      DO 10 I=1,5
10    ISUM=ISUM+K(I)  !***ISUM is uninitialized.
      TYPE 5, ISUM
      GOTO 1
      END
```

Upon executing the program, it can be seen that the first-round result is correct; but all subsequent results will be wrong. Such errors in a complicated program will be very difficult to detect just by looking at the run history and the result. The program will seem

to run normally, and the results may seem to be off, but not by much.

(3) Is there any negative, zero, or non-integer subscript? If there is, make sure it is used correctly.

FORTRAN-10 allows negative, zero, or non-integer subscripts. It is sometimes useful, for example, in the interpolation by integer increments. It is a dangerous practice. The negative or zero subscripts would make the array variable out of bound, unless they are specifically declared in the DIMENSION statement. The non-integer subscripts are always truncated (not rounded) into integers in assigning values.

(4) Is there any "off-by-one" error in the DO-loops, iterations, counters and indexes?

(5) If more than two variable alias share a common storage, such as by using an EQUIVALENCE statement, are these variables of the same type? If they are of different types, does the program contain any steps that store a value as one variable alias and later use it as a different alias?

(6) In coding a subprogram, have all parameter variables been explicitly declared with their types and dimensions?

(7) In coding a subprogram, are the dimension specifications of array variables consistent between the main program and the subprogram. If the subprogram contains an adjustable dimension, make sure that the original size of the array does not exceed the size of the array assigned within the subprogram, since the size of an array is not dynamically expandable.

(8) In using a subprogram, have all parameter-values been established at the subprogram CALL statement? In the CALL statement, does the subprogram contain a correct number of parameters in their correct sequence? Are the parameters identical in types to those defined in the subprogram?

(9) If a subprogram is called more than once in a program, the referenced variables in the subprogram should not be initialized by a DATA statement.

If a subprogram variable is initialized by a DATA statement, it will be correctly initialized when the subprogram is called for the first time. At the first conclusion of the subprogram execution, that variable value is altered by the subprogram computations. Now, in the same run, if the subprogram is called for the subsequent times, the initialized value of that variable will be the result left there in the previous call. Therefore in a program, if a subprogram is to be called more than once, the DATA statements in the subprogram should be used only to set constants, which are not altered in the subprogram execution. For initializing variables, use explicit assignment statements.

(10) In passing the values of parameters from the calling program to a subprogram or function, is the unit system consistent?

For example, the angle computation in the main program may be in degrees, but the angle parameter in the subprogram may require a value in radians.

(11) Some subprograms are written that the input variables are altered and returned as the output. Have the input variables been saved elsewhere (for example, by duplicating them with another variable name) if later computations require the same input variables?

In many subprogram construction, the output returned from the subprogram occupies the same storage area as the input to the subprogram to save storage. For example, the subprogram SUBROUTINE INVERS(N,A) may be written as a matrix inversion subprogram for a square matrix A of NxN size. To save storage, the A-array will accommodate the input A-matrix and return the A-inverse as the output. Thus, if the A-array is used later for other computations, the A-inverse will actually be used, unless the A-matrix is saved before calling the INVERS subprogram.

(12) Are the COMMON statements in all subprogram modules defined consistently?

(13) Are there any variables with very similar names, such as ROOT, ROOTS, ROOT1, ROOTX, etc?

The similar names are potential source of errors during transcription and entering the program. They tend to confuse the program and make the code inspection harder. While it is not necessarily an error, it is definitely a poor practice, because it sets up an error-prone situation.

(14) If a data file is referenced by more than one subprogram, do the different subprograms refer to the data structure consistently?

If a data file is an ASCII file, does it happen that it is referenced as an ASCII file at one time, but a binary file at another. Although both may contains numerical data, but the bit pattern interpretations will be different.

4.6 A Checklist for Computation Errors

(1) Beware of mixed-mode computations.

FORTRAN-10 allows mixed-mode computations. (See Section 3.6) However, one should be thoroughly familiar with the conversion rules as given in Table 3.5. For example, when you add a real constant to an integer constant, the result is a real constant. Thus, a statement as K=1-0.1 would yield K=0, but X=1-0.1 would yield X=0.9.

(2) Is it possible for divide-by-zero to occur?

For example, if you are writing a subprogram to solve for the roots of a quadratic equation: A*x**2+B*x+C=0. Have you included the possibility that "A" may be zero in the subprogram application?

(3) Does an overflow or underflow situation exist in the program?

Such situation may exist even though the execution seems to finish to a valid conclusion. If such situation exists, scaling or other manipulation may be necessary. For example, consider the statement:

```
X=(Y1*Y2*Y3*Y4)/(Z1*Z2*Z3*Z4)
```

If each of the values has a magnitude in the range of E+10, the multiplications in the numerator and the denominator would cause an overflow, even though the result may be within the range of the computer numbers. To avoid an intermediate overflow, rewrite the statement as:

```
X=Y1/Z1*Y2/Z2*Y3/Z3*Y4/Z4
```

(4) Have the computations considered the truncation and round-off errors in the decimal/binary number conversion?

For example, does the computation expect 100*0.1 to be 10? If a loop is initiated with a counter set at 0 and increment of 0.1, will the loop be terminated when 100 increments later the counter reaches 10? The following trivial program, seeming harmless, will actually create an endless loop:

```
      X=0.
10    IF(X.EQ.10.0)STOP
      X=X+0.1
      GOTO 10
      END
```

(5) Through the normal computation errors (roundoffs and truncations), can the value of a variable go beyond a meaningful range?

For exmaple, probability is never negative nor larger than 1; the argument for of an arc-sine function is never larger than 1; the argument for a logarithm function can never be non-positive; and the values of a rectified voltage cannot be negative. In such cases, upper and lower bound bias statements will be required in order that the computations subsequently will be meaningful.

(6) Always check the validity of integer divisions.

Integer arithmetics, except in division, always produces exact integer results, provided they are within range. Therefore, they are preferred in such operations as counting the loops and iterations, computation of subscripts and indexes, etc. When an integer division is encountered, the result is truncated rather than rounded. If rounded integer result of division is desired, then the expression of division should be "pre-rounded", as shown below:

```
IQ= INUM/IDENOM   replaced by:   IQ= (2*INUM+IDENOM)/(2*IDENOM)
```

Order of operations in the integer arithmetics is important too. The following two statements may produce different results:

```
IQ = (K1*K2)/(K3*K4)     versus    IQ = K1/K3*K2/K4
```

4.7 A Checklist for Logic Errors

Logic decisions in a FORTRAN-10 program are mainly made by decision (IF) statements with a subsequent object-action statement such as a transfer. Therefore, many common logic errors derive out of errors of incorrctly using the decision and object statements.

(1) In terminating a loop; is there any "off-by-one" error? This would result in an iterations with one loop too many or too few.

(2) Is the comparison done between two variables/constants of the same type? Is there any mixed mode comparison?

For example, is a real variable comparing with an integer variable or constant?

(3) Are logic connectives such as AND, OR, XOR correctly used?

Many people get confused on the difference between AND and OR in the precise logic meaning. In common English, we would say "the roots of the equation are 3 and 4"; while we actually mean logically "the roots of the equation are 3 or 4." Also, we often investigate conditions of "either, or", but fail to investigate the condition of "both", thus confusing the case of "OR" with the case of "EXCLUSIVE OR" or "XOR".

(4) Are the logic operators, such as .GT., .GE., .LT., and .LE., used correctly?

An error-prone situation is when the programmer decides to change the logic from, for example, .GE. to its inverse .LT. Have the actions based on the comparison decisions been changed correctly?

(5) How exhaustive is a comparison?

For example, suppose an integer variable is to have only two values, 1 or 2. In the comparison, can this value ever become other than 1 or 2, for example, by reading an input? If the value is not 1, is it automatically assumed to be 2? If so, you are really trying to distinguish 1's from non-1's. If the value is neither 1 nor 2, can the program handle the situation?

(6) In a multi-destination branching statement, such as the COMPUTED or ASSIGNED GO TO statement:

```
GO TO (N1,N2,...NK) K
GO TO K, (L1,L2, ...LK)
```

Can the variable K reach a value that exceeds the number of branching alternatives?

(7) Make certain that every loop, every subprogram, and every program eventually will terminate.

Having a STOP or RETURN statement at the end of a program is not an conclusive evidence that a program will terminate. The program may never reach the STOP statement. Design some informal proof or program walkthrough, that a program will terminate under all conceivable condtions.

(8) Is there any portion of the program that will never be executed?

For example, consider the statement:

```
IF(IQ.LT.0)CALL IQTEST(A,B,C,D)
```

If IQ never goes negative, the subprogram IQTEST would never be called. Is it an oversight? It calls for a detailed re-examination for that statement.

4.8 A Checklist for Input/Output Errors

(1) In the OPEN statement for a file, are all parameters given correctly? For parameters not stated in the OPEN statement, are the default values satisfactory for the file application? Can the file be shared with other users during the program execution?

(2) Have all files been properly opened before accessing them?

(3) If the opening of a file is likely to have problem, for example, to open a shared file, is there a "ERR=" parameter in the OPEN statement to handle the error situation?

(4) For every READ/WRITE statement, are the listed data perfectly consistent with the format specifications referred to by them?

(5) Even if the format specifications are consistent, is there any data truncation caused by the formats?

For example, if a 5-character alphanumberic variable in a program is outputed by a format of "A3", the variable is truncated to the leading three characters. Is this an oversight, or is it done with a purpose?

(6) In the format specifications for the WRITE statements, has the first column been reserved for the FORTRAN carriage-control characters? Therefore, in a typical output format, is the first column blank?

For example, a format of "5I1" for outputing (K(I),I=1,5) would cause unpredictable carriage movements and K(1) deleted in the output. The correct format should be "1X,5I1".

(7) If file reading is involved in a loop operation, the number of records to be read by the loops should not be more than the file contains. The READ statement should have a parameter "END=n" included to handle end-of-file situation.

Many loops are designed on terminating upon end-of-file condition. When the file is revised with more or less records, the loops will still be terminated correctly.

4.9 A Checklist for Program Readability

In order to do the code inspection, it is necessary that people must be able to read and understand the code in order to correct, modify, and debug it. Unfortunately, it is often easier to re-write someone else's program than to modify it.

Conventional languages use punctuation, indentations, paragraphing, ordering and spacing to improve readability. We should also use these practices to reduce the chance of misunderstanding. Analogy of good programming style can be drawn on a good writing style of conventional English langauage. Good style, of course, is a matter of individual opinion. What is a good style to one person may be too restrictive to another. Here we will include a checklist for good style, but one should realize that "good style" is subject to individual interpretation. The following shows a list of commonly accepted good programming styles:

(1) Write the comments as you code.

Details are fresh at coding time. Later, it will be difficult to remember what to be commented. A program with explanatory comments is much easier to read and debug.

(2) As blank lines are used in English to separate paragraphs, use blank lines to separate groups of statements. Also, use a blank line after an unconditional transfer (GO TO) statement to indicate a break in the program flow.

(3) Select names to increase readability.

```
Poor:      Z = X * Y
Better:    VOLT = AMPERE * OHM

Poor:      SUBROUTINE X001(X1,X2)
Better:    SUBROUTINE PRICE(COST,PROFIT)
```

(4) Use a standard rule to define abbreviations (such as for the variable names)

a. Initial letter must be present.
b. Consonants are more important than vowels.
c. Beginning of a word is more important than the end.
d. Add "I" or "X" prefix to the abbreviation to correct the data type.

(5) Arrange the name list in alphabetic order and in neat columns, especially if the list is long. Use tab keys.

```
Poor:      INTEGER ARRAY(128),PIXEL(128),QUOTE(5),TAU(128),WEIGHT,
          1SIGNAL,REQST,RESIST,LAB,GAMMA,BETA

Better:    INTEGER  ARRAY(128),     PIXEL(128),     QUOTE(5),
          1         TAU(128),      BETA,           GAMMA,
          2         LAB,           REQST,          SIGNAL,
          3         WEIGHT
```

(6) Sequence the statement labels so that the statements may be quickly located in a large program.

(7) Paragraphing a DO-loop and an IF-statement group. Use indentations.

```
Poor:      DO 20 I=1,1                    Better:    DO 20 I=1,10
           X(I)=0.0                                    X(I)=0.0
           DO 10 J=1,5                                 DO 10 J=1,5
           Y(I,J)=0.0                                    Y(I,J)=0.0
         10 CONTINUE                                 10  CONTINUE
         20 CONTINUE                                 20 CONTINUE

Poor:      IF(KODE.GT.3) GO TO 5          Better:    IF(KDOE.GT.3)GO TO 5
           CALL STEP1(A,B,C)                           CALL STEP1(A,B,C)
           CALL STEP2(A,B,C)                           CALL STEP2(A,B,C)
          5 CALL STEP3(A,B,C)                         5 CALL STEP3(A,B,C)
```

ON-LINE PROGRAM DEBUGGING BY DIAGNOSTIC REPORTS

After an exhaustive off-line debugging process, correct all detected errors, and we are now ready for the on-line processing.

For the on-line processing of FORTRAN program, the sequence of computer processing is first to compile the source program, then to load the object programs, and finally to execute. Errors may be detected at each stage by the system processors, and these errors are reported on the user's terminal as the error diagnostic messages. They are very helpful in identifying the errors discovered at each stage, and they will be discussed next in some details.

4.10 Compiler Diagnostics

The FORTRAN-10 compiler has an extensive error checking and diagnostic capability to diagnose and report the errors in the source program. The report includes such pertinent information as the line number of the malfunctioning statement in the source, reprinting of the statement, and a brief message describing and diagnosing the error.

There are two levels of error messages. A "warning message" indicates either an inconsistency or a tolerable minor error. The compilation will continue. The "fatal error message" indicates that the error will result in an incorrectly compiled object program if allowed to continue, and hence, the compiling is aborted. Thus, a warning message does not necessarily indicate an error, but often a tolerable bad or unsual practice. Consider the following two segments of program:

```
      DO 10  I=1,25                VOLTAGE=CURRENT/RESISTANCE
10 CALL SUB(A,B,I)                 TYPE 100, VOLTAGE
```

When the programs containing these segments are compiled, both will produce warning messages. The segment on the left will be objected by the compiler that the DO-loop index "I" is being passed to a subprogram, and therefore may be possibly altered upon its return. The segment on the right is objected by the compiler because the variable names contain more than six characters. In both cases compilation continue unless aborted by fatal errors down stream. For the segment shown on the right half side, the variable names are actually truncated into six-character names. Some programmers have a habit of using full variable names rather than their abbreviations so that the program is easier to read. This is a good practice so long as the risks are understood. For example, the variable names VOLTAGE1, VOLTAGE2, VOLTAGE3 will not be different variable names after being truncated to six characters, and all 3 variables will be treated by FORTRAN-10 as the variable VOLTAG. Therefore, only when the warnings are well understood, then we can ignore them if we are certain no error in the program is committed.

The diagnostics are derived by examining the error symptoms and concluding with a most probable diagnosis. Often, the symptoms are secondary or even tertiary effect from the original error. For example, if an array X(I) is missing in the DIMENSION statement, or if there is an error in that DIMENSION statement, the compiler would not recognize later that X(I) is an array variable. It would take it for granted that it is a function named X with an argument I. and this error will not be reported at the compiling stage. Therefore, while the compiler diagnostic reports are extremely helpful in

identifying the trouble, but one should not be lulled into a false security if the compiler reports no error.

The error message format is:

?FTNXXX LINE:n text or %FTNXXX LINE:n text

where:

?FTN = FORTRAN compiler message, fatal error
%FTN = FORTRAN compiler message, warning
XXX = 3-letter mnemonic code, meaning of which shown below
LINE:n = line number where error occurs in the source program
text = explanation of error

Mnemonic Codes for Fatal Errors and Warnings

The fatal error messages on the user's terminal are preceded by "?FTN", and nonfatal error warnings by "%FTN". Tables 4.2 and 4.3 show a selected and summarized subset of diagnostic messages, along with explanations and examples.

There are many program errors of the type which the compiler cannot detect. Some of the common errors of this type are as follows:

(1) A part of the program is missing.

(2) Branching the wrong way from an IF statement.

(3) Wrong FORMAT associated with I/O data.

(4) Incorrect dimension or unspecified dimension of array.

(5) Incorrect parameter types in a subprogram.

(6) Array subscript out-of-bound of DIMENSIONed size.

4.11 Run-Time Diagnostics

(1) The FOROTS diagnostics

In the FORTRAN-10 program execution, the tasks of interfacing between the user's object programs and the DEC-10 monitor during I/O operations are carried out by a processor called FOROTS (FORTRAN Object Time System). In addition to the main tasks, other capabilities include job initialization, core management, error-handling and reporting, file management, data formatting, mathematic library, user library, specialized applications package, overlay facilites and FORTRAN IV compatibility. It is specifically the "error-handling and reporting" aspects of the FOROTS that we will be dealing here.

Code	Message Text (All Upper Cases) and Explanations
ATL	ARRAY [name] TOO LARGE
	The core required to accommodate this array is larger than the user's maximum allocation. Statements with Errors: / Corrected Statements: DIMENSION X(100,100,100) / DIMENSION X(100,100,10)
AWN	ARRAY REFERENCE [name] HAS WRONG NUMBER OF SUBSCRIPTS.
	The array is defined to have more or fewer dimensions than the given reference. Statements with Errors: / Corrected Statements: DIMENSION X(5,5,5) / DIMENSION X(5,5,5) X(1,1)=1.0 / X(1,1,1)=1.0
CQL	NO CLOSING QUOTE IN LITERAL
	Literal constants should be enclosed in closed quotes. Statements with Errors: / Corrected Statements: KAR='NAME / KAR='NAME' 10 FORMAT(' X1=,F8.2) / 10 FORMAT(' X1=',F8.2)
DIA	DO INDEX VARIABLE [name] IS ALREADY ACTIVE.
	In any nest of DO loops, a given index variable may not be used for more than one loop. Statements with Errors: / Corrected Statements: DIMENSION X(10,10) / DIMENSION X(10,10) DO 5 I=1,10 / DO 5 I=1,10 DO 5 I=1,10 / DO 5 J=1,10 5 X(I,I)=0.0 / 5 X(I,J)=0.0
DID	CANNOT INITIALIZE A DUMMY PARAMETER IN DATA.
	Statements with Errors: / Corrected Statements: SUBROUTINE SUB(A,B) / SUBROUTINE SUB(A,B) DATA A/1.0/ / A=1.0
DSF	ARGUMENT [name] IS SAME AS FUNCTION NAME.
	Statements with Errors: / Corrected Statements: FUNCTION FUNC(FUNC1,FUNC) / FUNCTION FUNC(FUNC1,FUNC2)

DTI	THE DIMENSION OF [arrayname] MUST BE OF THE TYPE INTEGER.
	Statements with Errors: / Corrected Statements: SUBROUTINE SUB(X,Y) / SUBROUTINE SUB(X,K) DIMENSION X(Y) / DIMENSION X(K)
DVE	CANNOT USE DUMMY VARIABLE IN EQUIVALENCE.
	Statements with Errors: / Corrected Statements: SUBROUTINE SUB(X1,X2,X3) / SUBROUTINE SUB(X1,X2,X3) EQUIVALENCE (X1,X2) / ... ... / X2=X1
EID	ENTRY STATEMENT ILLEGAL INSIDE A DO LOOP.
EIM	ENTRY STATEMENT ILLEGAL IN MAIN PROGRAM.
ENF	LABEL [number] MUST REFER TO AN EXECUTABLE STATEMENT, NOT A FORMAT. Statements with Errors: / Corrected Statements: GO TO 10 / GO TO 11 ... / ... 10 FORMAT(F8.2) / 10 FORMAT(F8.2) X=1.0 / 11 X=1.0
FEE	FOUND [symbol] WHEN EXPECTING EITHER [symbol] OR A [symbol]. This is a general syntax error message. The compiler detects something wrong, but not quite sure about what's wrong.
FNE	LABEL [number] MUST REFER TO A FORMAT, NOT AN EXECUTABLE STATEMENT. Statements with Errors: / Corrected Statements: 9 X=1.0 / 9 X=1.0 WRITE(6,9)X / WRITE(6,10)X 10 FORMAT(F8.2)
FWE	FOUND [symbol] WHEN EXPECTING [symbol]. This is another general purpose syntax error message. Statements with Errors: / Corrected Statements: X=RA*(F1+F2 / X=RA*(F1+F2)
IAC	ILLEGAL ASCII CHARACTER [character] IN SOURCE. Sometimes, a non-print ASCII character may be inadvertently entered in the source. Since it is not printed out, it may not be easily detectable in the proof reading process.

IAL	INCORRECT ARGUMENT TYPE FOR LIBRARY FUNCTION [name].
	Statements with Errors: / Corrected Statements: X=SIN(I) / XI=I X=SIN(XI)
IDN	DO LOOP AT LINE: [number] IS ILLEGALLY NESTED.
	The program attempts to terminate a DO loop before terminating one or more loops defined after the given one. Statements with Errors: / Corrected Statements: DO 10 I=1,10 / DO 10 I=1,10 X(I)=0.0 / X(I)=0.0 DO 20 J=1,5 / DO 20 J=1,5 Y(I,J)=0.0 / Y(I,J)=0.0 10 CONTINUE / 20 CONTINUE 20 CONTINUE / 10 CONTINUE
ILF	ILLEGAL STATEMENT AFTER LOGICAL IF.

	Two types of statements may not be the objective statement to a logical IF. One is a DO-statement, and the other is another logical IF statement.
	Statements with Errors: / Corrected Statements:
	IF(K.EQ.1)DO 10 I=1,10 / IF(K.EQ.1)GO TO 5
	... / 5 DO 10 I=1,10
	IF(I.EQ.1)IF(J.EQ.2) / IF((I.EQ.1).AND.(J.EQ.2))
	1 GO TO 5 / 1 GO TO 5

ISD	ILLEGAL SUBSCRIPT EXPRESSION IN DATA STATEMENT.
	Subscript expressions may be formed only with implicit DO indices and constants combined with +, −, *, or /. Statements with Errors: / Corrected Statements: DATA (X(I**2),I=1,5)/5*0.0/ / DATA (X(I*I),I=1,5)/5*0.0/
ISN	[symbolname] IS NOT [symboltype].
	The symbol cannot be used in the attempted manner. For example, a variable and a function cannot share the same name.

IXM	ILLEGAL MIXED MODE ARITHMETIC.
	Complex and double precision operands cannot appear in the same expression.
LAD	LABEL [number] ALREADY DEFINED AT LINE: [number]
LNI	LIST DIRECTED I/O WITH NO I/O LIST.
NIO	NAMELIST DIRECTED I/O WITH I/O LIST.
NIU	NON-INTEGER UNIT IN I/O STATEMENT.
NLF	WRONG NUMBER OF ARGUMENTS FOR LIBRARY FUNCTION [name].
NNF	NO STATEMENT NUMBER ON FORMAT.
NRC	STATEMENT NOT RECOGNIZED.
OPW	OPEN PARAMETER [name] IS OF WRONG TYPE.
	Statements with Errors: / Corrected Statements: REAL K / INTEGER K OPEN(UNIT=K,FILE='X.DAT') / OPEN(UNIT=K,FILE='X.DAT')
PIC	THE DO PARAMETERS OF [index name] MUST BE INTEGER CONSTANTS.
PTL	PROGRAM TOO LARGE.
RIC	COMPLEX CONSTANT CANNOT BE USED TO REPRESENT THE REAL OR IMAGINARY PART OF A COMPLEX CONSTANT.
SOR	SUBSCRIPT OUT OF RANGE.
UMP	UNMATCHED PARENTHESES.
USI	[symbol type] [symbol name] USED INCORRECTLY
VSO	VARIABLE DIMENSION ALLOWED IN SUBPROGRAMS ONLY.

Note: In the error message text, the names, numbers, characters enclosed in square brackets correspond to the actual parameters used in the source program.

Table 4.2 Mnemonic Codes for Fatal Compiling Errors

Code	Message Text (All Upper Cases) and Explanations
CUO	CONSTANT UNDERFLOW OR OVERFLOW
DIM	POSSIBLE DO INDEX MODIFED INSIDE LOOP.
	If a DO loop index is involved in an arithmetic expression, the program may be compiled incorrectly. The compiler will translate the DO loop index such that the number of iterations is calculated at the beginning of the loop and is never affected by modification of the index within the loop. Statements with Errors: / Corrected Statements: DO 10 I=1,10 / DO 10 I=1,10 10 CALL READRA(1,I,KOL) / IX=I 10 CALL READRA(1,IX,KOL) DO 10 I=1,10 / not allowed to alter the index IF(I.GE.3)I=I+1 etc
DXB	DATA STAEMENT EXCEEDS BOUNDS OF ARRAY .
FMR	MULTIPLE RETURNS DEFINED IN A FUNCTION.
ICC	ILLEGAL CHARACTER, CONTINUATION FIELD OF INITIAL LINE.
	Continuation lines cannot follow comment lines.
ICD	INACCESSIBLE CODE. STATEMENT DELETED.
	Such code may be an oversight, or may be a logic error. Check the flow chart on flow logic.
ICS	ILLEGAL CHARACTER IN LINE SEQ#.
IDN	OPT - ILLEGAL DO NESTING - OPTIMIZATION DISCONTINUED.
	The compiler contains a code optimizer. When a question arises in the source program, the optimizer is bypassed and compiler does straight translation.
IFL	OPT - INFINITE DOOP - OPTIMIZATION DISCONTINUED.
	The compiler contains a code optimizer. When a question arises in the source program, the optimizer is bypassed and compiler does straight translation.
LID	IDENTIFIER [name] MORE THAN SIX CHARACTER
	Statements with Errors: / Corrected Statements: VOLTAGE=CURRENT/RESISTANCE / Actual statement compiled: VOLTAG=CURREN/RESIST
MVC	NUMBER OF VARIABLE DOES NOT EQUAL THE NUMBERS OF CONSTANTS IN DATA STATEMENT.
NED	NO END STATEMENT IN THE PROGRAM.
NOF	NO OUTPUT FILE GIVEN.
SOD	[name] STATEMENT OUT OF ORDER
VNI	OPT - VARIABLE [name] IS NOT INITIALIZED.

Tab e 4.3 Mnemonic Codes for Compiler Warning Messages

The errors detected at run-time by FOROTS are reported using a format of:

%FRSXXX text

where %FRS = FOROTS error diagnostic report prefix
XXX = 3-letter code as defined in Table 4.4
text = error message

CODE	Expalantions
APR	Arithmetic fault errors, generated in calculations.
DAT	Data errors, generated in data conversion during an I/O operation.
DEV	Device error, generated by I/O hardware errors.
LIB	Library function errors.
OPN	File OPEN/CLOSE errors.
SYS	System errors, generated internally by FOROTS.

Table 4.4 Mnemonic Codes for FOROTS Run-time Errors

(2) Error traceback report

In addition to the FOROTS diagnostics, an error traceback report will also be printed on the user's terminal to aid him to locate the trouble. To fully utilize the FOROTS traceback, use the error report in conjunction with the compiler listing of the source. For that reason, let us follow a debugging case history to identify an error.

Example: We will take the Newton-Raphson method example, but insert an out-of-bound error in it so that it cannot be detected during compiling time. Then write a simple main program to call that subroutine. These source programs are stored as one file TEST.FOR. The program listing is shown below along with the line sequence numbers:

```
00001           CALL NEWTON(1.,-11.,9.,8.,20.,50,3,ROOT)
00002           TYPE 10, ROOT
00003   10      FORMAT(' ROOT = ', F8.2)
00004           END

00001   *
00002           SUBROUTINE NEWTON(A,B,C,D,E,ITER,NPLACE,ROOT)
00003           DIMENSION X(2)
00004           X(1)=-B/A
00005           ERROR=10.**(-NPLACE)
00006           F1=FUNCP(A,B,C,D,E,X(1))
00007           IF(F1.EQ.0.)X(1)=2.*X(1)
00008           DO 10 I=1,ITER
00009           X(2)=X(1)-FUNC(A,B,C,D,E,X(10))/FUNCP(A,B,C,D,X(1))
00010           IF(ABS((X(2)-X(1))/X(1)).LE.ERROR)GOTO 20
00011           X(1)=X(2)
00012   10      CONTINUE
00013           WRITE(6,100)ITER
00014   100     FORMAT(/'*** NO CONVERGENCE WITHIN',I5,' ITERATIONS.')
00015           RETURN
00016   20      ROOT=X(2)
00017           RETURN
00018           END

00001   *
00002           FUNCTION FUNC(A,B,C,D,E,X)
00003           FUNC=A*X**4+B*X**3+C*X**2+D*X+E
00004           RETURN
00005           END

00001   *
00002           FUNCTION FUNCP(A,B,C,D,X)
00003           FUNCP=4.*A*X**3+3.*B*X**2+2.*C*X+D
00004           RETURN
00005           END
```

Note that the line No.00008 of the subroutine NEWTON has an out-of-bound error. X(1) is incorrectly entered as X(10).

Upon a compiling command, the following printout was obtained:

```
.COMPILE TEST.FOR
FORTRAN 5A(621): TEST.FOR
MAIN.   OCTAL PROG SIZE=53
FORTRAN 5A(621): NEWTON.FOR
NEWTON  OCTAL PROG SIZE=211
FUNC    OCTAL PROG SIZE=52
FUNCP   OCTAL PROG=55
```

Therefore, it appears that compiling for the main program and three subprograms were successful, because there is no error message. Next the program is executed:

```
.EXECUTE TEST.FOR
LINK:   loading
[LNKXCT TEST execution]

%FRSAPR Floating overflow at FUNC+16[457]
FUNC[441] called from NEWTON+47[322] with 6 args of type F,F,F,F,F,F
NEWTON[253] called from MAIN.+4[161] with 8 args of type F,F,F,F,F,I,I,F


(The same report is repeated several times)

%FRSDAT Output field width overflow
Unit=-1 TTY:/ACCESS=SEQINO/MODE=ASCII
Input record = 0; Output record = 1;
(' ROOT =',F8.2)
IOLST.[404320] called from MAIN.+10[165] with no args

ROOT = ********

End of Execution    FOROTS 5B(1001)
CPU time: 0.06  Elapsed time: 0.07

No. of          Error
Errors          Type
1               Output field width overflow
76              Floating overflow

EXIT
```

It is at this point that many people are overwhelmed. Actually, to a FORTRAN user, there are much useless information he can simply ignore. For example, all numbers in square brackets are actual core addresses, and they are useless to a FORTRAN user. Therefore, focus your attention on the expressions having the form PROGRAMNAME+NUMBER, such as NEWTON+47, which are memory locations in relative addresses. PROGRAMNAME is the base address of the program unit, and NUMBER is the offset indicating the relative address. So, let us interpret the trace report without the core addresses:

"%FRSAPR Floating overflow at FUNC+16"
Meaning: Arithmetic error (APR) of floating point overflow occured in the program unit FUNC, and at a location of relative address 16. If we have a location map of relative address of FUNC statements, the offending statement may be quickly identified. This relative address map is on the compiler listing.

"FUNC called from NEWTON+47 with 6 args of type F,F,F,F,F,F"
Meaning: The offending subprogram is called by the statement in the subprogram NEWTON with a relative address of 47. Again a compiler listing will quickly identify the calling statement.

"NEWTON called from MAIN.+4 with 8 args of the type F,F,F,F,F,I,I,F"
Meaning: The subprogram NEWTON was called by a statement in the MAIN unit that has a relative address of 4.

The other error report passage indicates the error in I/O processing, purported to be out of range of the assigned format. The format is printed to aid the identification.

(3) Use compiler listing to locate the error

A. How to get a compiler listing

To get a compiler listing, use a switch /LIST when applying the command for COMPILE, LOAD or EXECUTE. However, if the source program has been previously compiled and a valid REL file already exists, delete that REL file first. Otherwise, no new compiling will be done. To compile (or load, or execute) and produce a compiler listing, use the command:

.COMPILE FLNAME.FOR/LIST

In addition to producing a FLNAME.REL file, this command will also create a compiler listing file stored as FLNAME.LST. Carrying on the example started in the previous part, a compiler listing is produced by a command of "COMPILE TEST.FOR/LIST", and the result is shown on the next three pages.

Reproduction of Compiler Listings

```
MAIN.   TEST.FOR        FORTRAN V.5A(621) /KI/L 10-OCT-80       9:51    PAGE 1

00001           CALL NEWTON(1.,-11.,9.,8.,20.,50,3,ROOT)  } Listing
00002           TYPE 10, ROOT                             } of
00003   10      FORMAT(' ROOT = ', F8.2)                  } Main
00004           END                                       } Program

SUBPROGRAMS CALLED  }  Called
                    }  one subprogram
NEWTON              }

SCALARS AND ARRAYS [ "*" NO EXPLICIT DECLARATION - "%" NOT REFERENCED - " " SUBS
CRIPTED ]           }  Need
                    }  1 space
*ROOT       1 R     }  for data

TEMPORARIES  -------------------------- No temporary variable

LINE NUMBER/OCTAL LOCATION MAP  --- Line #2 stored in Locations 5-10 (octal)

             0        1        2        3        4        5        6        7        8        9
------:-------------------------------------------------------------------------------------------
      :
00000 :               3        5        -       11

MAIN.   OCTAL PROG SIZE=53  [ SCALARS/ARRAYS=1 + FORMATS=4 + TEMPS/CONS=7 +
CODE=
13 + ARGS=24 ]
[ NO ERRORS DETECTED ]
```

```
00001    *
00002              SUBROUTINE NEWTON(A,B,C,D,E,ITER,NPLACE,ROOT)
00003              DIMENSION X(2)
00004              X(1)=-B/A
00005              ERROR=10.**(-NPLACE)
00006              F1=FUNCP(A,B,C,D,E,X(1))
00007              IF(F1.EQ.0.)X(1)=2.*X(1)
00008              DO 10 I=1,ITER
00009              X(2)=X(1)-FUNC(A,B,C,D,E,X(10))/FUNCP(A,B,C,D,X(1))
00010              IF(ABS((X(2)-X(1))/X(1)).LE.ERROR)GOTO 20
00011              X(1)=X(2)
00012    10        CONTINUE
00013              WRITE(6,100)ITER
00014    100       FORMAT(/'*** NO CONVERGENCE WITHIN',I5,' ITERATIONS.')
00015              RETURN
00016    20        ROOT=X(2)
00017              RETURN
00018              END

SUBPROGRAMS CALLED

ABS.     FUNCP    FUNC
SCALARS AND ARRAYS [ "*" NO EXPLICIT DECLARATION - "%" NOT REFERENCED - " " SUBS
CRIPTED ]

*E            1 R        *B            2 R        *ITER         3 I
*D            4 R        *A            5 R
*ROOT         6 R        *ERROR        7 R        *NPLACE      10 I
 X           11 R        *I           13 I
*F1          14 R        *C           15 R

TEMPORARIES

 .S0000      30 I         .A0016      31 R         .Q0000       32 I

LINE NUMBER/OCTAL LOCATION MAP

      :  0       1       2       3       4       5       6       7       8       9
------:-------------------------------------------------------------------------------
      :
00000 :          -       0       -      21      24      31      34      40
43
00010 : 54      62      64      66       -      72      73       -      75

NEWTON OCTAL PROG SIZE=211 [SCALARS/ARRAYS=15 + FORMATS=12 + TEMPS/CONS=6 + CODE
=114 + ARGS=40 ]
[ NO ERRORS DETECTED ]

00001    *
00002              FUNCTION FUNC(A,B,C,D,E,X)
00003              FUNC=A*X**4+B*X**3+C*X**2+D*X+E
00004              RETURN
00005              END

SUBPROGRAMS CALLED
```

Listing of Program Unit "NEWTON"

3 Subprograms called

Variables assigned

Compiler generated (Ignore them)

For example, Line #13 code of "WRITE(6,100) ITER" is stored at locations 66 - 71.

```
SCALARS AND ARRAYS [ "*" NO EXPLICIT DECLARATION - "%" NOT REFERENCED - " " SUBS
CRIPTED ]

*E          1 R       *B          2 R       *D          3 R
*A          4 R       *X          5 R
*C          6 R       *FUNC       7 R

TEMPORARIES

 .A0002    10 R        .A0003    11 R

LINE NUMBER/OCTAL LOCATION MAP

      : 0       1       2       3       4       5       6       7       8       9
------:-------------------------------------------------------------------------
      :
00000 :         -       0      16       -      35

FUNC    OCTAL PROG SIZE=52 [SCALARS/ARRAYS=7 + TEMPS/CONS=2 + CODE=41 + ARGS=0]
[ NO ERRORS DETECTED ]
```

```
00001    *
00002            FUNCTION FUNCP(A,B,C,D,X)
00003            FUNCP=4.*A*X**3+3.*B*X**2+2.*C*X+D
00004            RETURN
00005            END

SUBPROGRAMS CALLED

SCALARS AND ARRAYS [ "*" NO EXPLICIT DECLARATION - "%" NOT REFERENCED - " " SUBS
CRIPTED ]

*B          1 R       *D          2 R       *A          3 R
*X          4 R       *FUNCP      5 R
*C          6 R

TEMPORARIES

 .A0002     7 R        .A0003    10 R        .A0004    11 R        .A0005
  12 R

LINE NUMBER/OCTAL LOCATION MAP

      : 0       1       2       3       4       5       6       7       8       9
------:-------------------------------------------------------------------------
      :
00000 :         -       0      16       -      35

FUNCP   OCTAL PROG SIZE=55 [SCALARS/ARRAYS=6 + TEMPS/CONS=4 + CODE=43 + ARGS=0]
[ NO ERRORS DETECTED ]
```

Each program (the main program, each subroutine, each function) has a complete compiler listing that contains the following parts:

a. The program listing. The program unit is listed with line sequence numbers assigned in the exact way the source program is prepared. All comment lines, continuation lines and blank lines will all be assigned with unique line sequence numbers. They are decimal numbers.

b. List of subprogram names called by this unit. Library functions are identified by names followed by a period; user functions with names only. If there is an unfamiliar function or subroutine name in the list, it identifies a possible error. The subprogram name may have been misspelled, or an array variable did not get DIMENSIONed.

c. "Scalars and Arrays". These are all variable names used in the program unit. Each variable name listing has a format of:

xNAME n # Type

where: x = type declaration code:

Code	Meaning
*	variable not declared explicitly
%	type declared but variable not used
blank	dimensioned as conventional type

NAME = variable name

n = relative address in octal number

= indicating this is an array

datatype = data type codes:

Code	Meaning
I	integer
R	real
C	complex
D	double precision
L	logical

By going through the list and cross-checking the program, mis-typing of data can be quickly identified.

d. "Temporaries". These are temporary variables generated by the compiler. Ignore this part; they are useless to FORTRAN users.

e. Location map. The map is printed in a matrix form. The row and column headings together form the line number, and the matrix element value is the first address (relative) of instruction codes translated from that line. Let us reproduce the map of the NEWTON subroutine of the last part:

```
LINE NUMBER/OCTAL LOCATION MAP

       : 0      1      2      3      4      5      6      7      8      9
-------:-------------------------------------------------------------------
       :
->00000 :        -      0      -     21     24     31     34     40     43
 00010 : 54     62     64     66      -     72     73      -     75
```

From this map, we can easily determine the core address assignment (in relative addresses) for every line of the program unit. For example, in the program unit NEWTON, the core assignment in relative addresses is as follows:

```
Line #: 4        Core addresses:  21-23 octal
        5                        24-30
        6                        31-33
        7                        34-37
        8                        40-42
      etc                        etc
```

f. Program size. The program size is broken into its components, and their sum is the total program unit size in octal number. Program size=53 means octal 53 DEC-10 words, or decimal 43 DEC-10 words.

B. How to use the compiler listing to locate errors

Once the relative address is known, the line number and the statement can be quickly identified.

Returning to the last example, FUNC+16 identifies Line #3 of FUNC; NEWTON+47 identifies Line #9 of NEWTON; and MAIN.+4 identifies Line #1 of the main program. The conclusion of the tracing report is:

The error is generated by Line No. 3 of the FUNC program, which is called by Line No. 9 of the NEWTON program, which in turn is called by Line No.1 of the main program.

(4) The ERRSNS subroutine

In addition to the above run-time FOROTS error-reporting facilities, there is a FORTRAN library subroutine ERRSNS that may be called by the "ERR=" parameter of READ, WRITE, OPEN and CLOSE statements in the form:

```
      OPEN(UNIT=1,FILE='INPUT.DAT',ERR=9999)
       ...
9999  CALL ERRSNS(I,J)
      TYPE 9998, I,J
9998  FORMAT(' READ/WRITE OR OPEN/CLOSE ERROR CODES:', 2I4)
      STOP  !or other error handling steps
```

The meanings of two integers "I" and "J" returned by the error-report subroutine ERRSNS are tabulated in Table 4.5:

I	J	Explanation
0	0 101	no error completion with no error
23	312	Backspace illegal for device
24	308	Reaching end of file during READ
25	opt	invalid record number
26	opt	Sequential file used as a random access
28	opt 254 262 268	CLOSE error Rename file already exists No room, quota exceeded Cannot delete or rename a file
29	opt 250	no such file File not found
30	opt 240 242 245 248 249 251 253	OPEN failure Record length spec missing for random access Too many devices opened (Max: 15) Device not available Illegal access for device Illegal MODE or MODE switch No such PPN File being modified, not available now
31	opt 315	Mixed access modes Random access file used as sequential
32	239	Invalid logical unit number
39	opt	READ error
45	opt	OPEN statement keyword error
47	263	Attempt to WRITE on READ-only file
62	opt 301 314	FORMAT syntax error Illegal character in FORMAT Missing width for A- or R-FORMAT on input

Table 4.5 Selected Numeric Codes of Error Report Subroutine ERRSNS

4.12 Dimension Out-of Bound Errors

Debugging frustration usually is derived from two causes: (1) the user is not familiar with the meaning of the error message, and (2) the user relies on the error diagnostics blindly. The first cause may be easily remedied. The material presented so far should be helpful in resolving the first cause somewhat. The second cause will be difficult to remedy, unfortunately. The error diagnostics are based on a failure symptom that may be a secondary or indirect effect of the original offender. By far the most frequent culprit in raising the programmer frustration is the dimension out-of-bound type of errors.

When a DIMENSION statement is specified in the source program, the compiler records the number of reserved storage locations, translate the source into the object codes, and reserve necessary argument storage and I/O buffers. These storages form a contiguous entity and later the LOADER will try to find a contiguous space in the core to fit it. There is no policing at the execution time to see that an array will not go beyond its assigned space.

As a result of compiling, each array is identified by a base address where the first element of the array will be stored. If the X-array has a base address at "ADDX", then X(1) is stored at ADDX, X(2) at (ADDX+1), X(3) at (ADDX+2), and so on, and X(k) at (ADDX+k-1). The quantity (k-1) is called the OFFSET, and therefore, the address of an element of array can be identified by computing (BASE+OFFSET). Access to an element in the array is performed in this manner. The OFFSET of a multi-dimensioal array is computed on a linear array basis. Naturally, for correct computations of a K-element array, the OFFSET should not be larger than (K-1) and should never be negative. Unfortunately, at the run-time, the OFFSET is not checked with the array size K. Hence, a dimension out-of-bound error cannot be detected per se at run-time but will be detected by the damage, if any, caused by it.

In a typical core storage for a program execution, in contiguous order are the storage areas for: data area, formats, temporarily generated codes, instruction codes, subprogram argument codes, I/O buffer areas, etc. If an array located in the data area is too large for its assigned locations, the surplus elements will go to other data storages, or into the instruction codes, or into the I/O areas, all dependent on the OFFSET calculation. Naturally, in this process, the information of the invaded area are altered and the result becomes unpredictable. Thus when the invaded area information is used for a subsequent execution, anything can happen.

Suppose the following is a segment of a FORTRAN program:

```
      DIMENSION X(100),Y(10)
      ...
      DO 5 I=1,150
    5 X(I)=0.0
      ...
```

Let us now analyze the consequences.

(1) The extra X-array elements may only alter other data storages, and this alternation will makes the result of computation invalid. On the other hand, the remaining computations may not need the affected values, and the results may be correct. Therefore, such a program may sometimes produce correct results, sometimes not. But the program will successfully run to a completion. This is often a case where a programmer blames on the "temper" of the machine.

(2) The extra values may invade the instruction code area and alter the contents. The result will be unpredictable. Coincidentally, the alteration may change a code to another perfectly legitimate code, and the program execution will take on a new and strange direction. Most likely, the alteration will result in a non-executable instruction, and an error message to that effect comes back to the puzzled user.

(3) The extra elements may alter some subscript- or index-calculation code in such way that a very large subscript value is obtained. Its OFFSET may be so large that it exceeds the boundary of the user's core allocation. Imagine how the user feels when he sees an error message of "NEED MORE CORE" for his short program.

(4) If the extra values invade the I/O buffer area where input/output data are formulated according to a FORMAT, an error message of "AN ILLEGAL CHARACTER IN THE FORMAT" will send the user on a unproductive wild goose chase.

Thus, while the out-of-bound error is one of the most frequently committed errors, its detection is far from obvious.

Because of the high incidence of such errors, an effectvie strategy may be as follows:

(1) After an error diagnostic message is received, the source program is checked as reported by the diagnosis. Often, these circumstantial evidences fail to unearth the real trouble. Then, always suspect first there is an out-of-bound error.*

(2) Delete the compiled REL file, and re-execute the source file now with a /DEBUG:BOUNDS switch. Note that the out-of-bounds error cannot be detected just by re-compiling even with the DEBUG:BOUNDS switch. That switch will perform the debugging only when the program is executed.

(3) Insert several core-occupying but meaningless statements in the suspected source program, for example:

```
      DIMENSION YYYY(100)
      DO 9999 I=1,100
 9999 YYYY(I)=1.2345
```

The purpose is simply to shift the core address assignments. This revised program is executed again, and another error report is obtained. If a different set of errors is reported this time, it is highly probable that an out-of-bound dimension error exists.

(4) When an out-of-bound error is suspected or confirmed, catch that error first and ignore other reported errors for the time being. Very probably, many secondary errors will be automatically corrected once the primary error is caught and corrected.

---

*The second most probable error source is an uninitialized or unset variable in computations.

ON-LINE DEBUGGING BY CONDITIONAL COMPILING

4.13 The D-Statement

In the on-line debugging of a FORTRAN-10 program, we are mainly trying to test the program to see (1) if the flow logic goes according to the plan, and (2) how the computation of data is going. The first means a tracing process, and the second means to inspect the values of variables at different stages of their computations. Both of these may be realized by output statements inserted at strategic places with formats that indicate a tracing and/or data inspection. For example, use output formats as illustrated below:

```
FORMAT(' I REACH POINT A; ALL IS FINE.')
FORMAT(' I REACH POINT B; ALL IS WELL.')
FORMAT(' AT POINT P; X-ARRAY DATA ARE: ',data-format ...)
```

But there are several problems. Once these statements are inserted, it will be difficult to distinguish them from the rest. When debugging is completed, we want to remove these extra statements, and that will be a tedious and error-prone process. Very likely, new bugs would be created by such procedure.

FORTRAN-10 allows a D-type statement that has a letter "D" in the first column. Let the program file name be FLNAME.FOR. If the following command is given:

```
EXECUTE FLANEM.FOR(I)
```

where "(I)" is a compiler switch to include D-statements, the program with all its regular and D-statements will be compiled and executed. The same program, if compiled and executed without the (I) switch, will treat the D-statements as comments and therefore ignore them.

After debugging is completed, delete the REL file, and recompile without the (I) switch.

Example: We will again use the same TEST.FOR but add some D-statements as shown below:

```
      CALL NEWTON(1.,-11.,9.,8.,20.,50,3,ROOT)
      TYPE 10, ROOT
10    FORMAT(' ROOT = ', F8.2)
      END
*
      SUBROUTINE NEWTON(A,B,C,D,E,ITER,NPLACE,ROOT)
      DIMENSION X(2)
      X(1)=-B/A
      ERROR=10.**(-NPLACE)
      F1=FUNCP(A,B,C,D,E,X(1))
      IF(F1.EQ.0.)X(1)=2.*X(1)
      DO 10 I=1,ITER
D     WRITE(6,9999) I,X(1)
D9999 FORMAT(/' ITERATION=',I3,'   X(1)=',E12.4)
      X(2)=X(1)-FUNC(A,B,C,D,E,X(10))/FUNCP(A,B,C,D,X(1))
D     WRITE(6,9998) I,X(2)
D9998 FORMAT(' AFTER ITERATION ',I3, ' X(2)=',E12.4)
```

```
      IF(ABS((X(2)-X(1))/X(1)).LE.ERROR)GOTO 20
      X(1)=X(2)
10    CONTINUE
     WRITE(6,100)ITER
100   FORMAT(/'*** NO CONVERGENCE WITHIN',I5,' ITERATIONS.')
      RETURN
20    ROOT=X(2)
      RETURN
      END
*
      FUNCTION FUNC(A,B,C,D,E,X)
D     WRITE(6,9999) A,B,C,D,E,X
D9999 FORMAT(' PARAMETERS PASSED INTO THE FUNC PROGRAM:'/
D    1  '  EQUATION COEFFICIENTS ARE:' /12X,5E12.4/
D    2  '  X(1) = ',E12.4)
      FUNC=A*X**4+B*X**3+C*X**2+D*X+E
D     WRITE(6,9998)FUNC
D9998 FORMAT(' RETURNED FROM FUNC, FUNC=',E12.4)
      RETURN
      END
*
      FUNCTION FUNCP(A,B,C,D,X)
D     WRITE(6,9999) A,B,C,D,E,X
D9999 FORMAT(' PARAMETERS PASSED INTO THE FUNCP PROGRAM:'/
D    1  '  EQUATION COEFFICIENTS ARE:' ,5E12.4/
D    2  '  X(1) = ',E12.4)
      FUNCP=4.*A*X**3+3.*B*X**2+2.*C*X+D
D     WRITE(6,9998)FUNCP
D9998 FORMAT(' RETURNED FROM FUNCP, FUNCP=',E12.4)
      RETURN
      END
```

Next, an EXECUTE command with the switch (I) is applied. Now a curious thing happens. The example program was aborted during the regular run but it runs to a completion with the D-statements. As the D-statements are merely output and format statements, suspiction should be raised here that the trouble is illegal data located out of bound. The terminal printout is included here along with some analysis:

```
 PARAMETERS PASSED INTO THE FUNCP PROGRAM:
  EQUATION COEFFICIENTS ARE:  0.1000E+01 -0.1100E+02  0.9000E+01  0.8000E+01
  X(1) =   0.2000E+02
 RETURNED FROM FUNCP, FUNCP=  0.1917E+05

 ITERATION=  1   X(1)=  0.1100E+02
 PARAMETERS PASSED INTO THE FUNCP PROGRAM:
  EQUATION COEFFICIENTS ARE:  0.1000E+01 -0.1100E+02  0.9000E+01  0.8000E+01
  X(1) =   0.1100E+02
 RETURNED FROM FUNCP, FUNCP=  0.1537E+04
 PARAMETERS PASSED INTO THE FUNC PROGRAM:
  EQUATION COEFFICIENTS ARE:
          0.1000E+01 -0.1100E+02  0.9000E+01  0.8000E+01  0.2000E+02
  X(1) =   0.1596E-01
 RETURNED FROM FUNC, FUNC=  0.2013E+02
 AFTER ITERATION   1 X(2)=  0.1099E+02

 ITERATION=  2   X(1)=  0.1099E+02
 PARAMETERS PASSED INTO THE FUNCP PROGRAM:
  EQUATION COEFFICIENTS ARE:  0.1000E+01 -0.1100E+02  0.9000E+01  0.8000E+01
  X(1) =   0.1099E+02
```

```
 RETURNED FROM FUNCP, FUNCP=  0.1527E+04
 PARAMETERS PASSED INTO THE FUNC PROGRAM:
  EQUATION COEFFICIENTS ARE:
             0.1000E+01 -0.1100E+02  0.9000E+01  0.8000E+01  0.2000E+02
  X(1) =    0.1596E-01
 RETURNED FROM FUNC, FUNC=  0.2013E+02
 AFTER ITERATION    2 X(2)=  0.1097E+02

      (SEVERAL MORE PAGES OF THIS)

 ROOT =    0.00


End of execution     FOROTS 5B(1001)
CPU time: 0.12  Elapsed time: 1.58
EXIT
```

From the analysis shown, the trouble is attributed to the fact that X(1) is passed into two subprograms FUNC and FUNCP as two different values! Armed with this information, the parameter lists of FUNC and FUNCP are examined, and the trouble can be identified quickly.

Note that the debug run actually produced an incorrect result of ROOT=0.00! The kind of errors causing computer jobs abortion is actually the safe kind. The really dangerous kind is an error causing not obviously incorrect results. As the popular saying goes, "Garbage in; garbage out," or "GIGO," but beware of camouflaged garbage.

ON-LINE DEBUGGING BY TRACING AIDS

Two subprograms are available in the FORTRAN-10 library for the tracing operations. One is to traceback at a specifed point; the other is to trace the flow in general. They are now presented next.

4.14 The TRACE Subprogram

The TRACE subprogram may be used as a subroutine without a dummy argument or a function with a dummy argument. when this subprogram is called at one point in the program execution, a printout of traceback will be produced at the user's terminal. The TRACE program is also automatically invoked in response to a PAUSE 'T' statement.

Example: Placing a TRACE call between Line #8 and #9 of the NEWTON subroutine as shown on the compiler listing of Section 4.11, the following printout was received on the terminal:

```
TRACE.[415615] called from NEWTON+45[322] with no args
NEWTON[255] called from MAIN.+4[161] with 8 args of type F,F,F,F,F,I,I,F
```

(many other lines)

Refer to Section 4.11 on how to read these tracings.

4.15 The MSFLVL Subroutine

The MSGLVL subroutine is another FORTRAN-10 library program which does a dynamic tracing of subprogram calls and returns, and labeled statements. It can be used to trace an entire program or a portion of it.

The format of the subroutine is:

```
CALL   MSGLVL(N)
```

where N is defined as:

```
N = 1    no tracing, used to turn off tracing.
N = 2    print out only subprogram calls and returns
N = 3    N=2 case plus all labeled statement tracings
```

Thus, at a point in the program where we want to start the tracing, we insert a statement "CALL MSGLVL(2)" or "CALL MSGLVL(3)". Then at the point where we want to turn it off, we insert a statement "CALL MSGLVL(1)".

Users of this routine should be warned that if this routine is called by a user's subroutine, the tracing is not automatically turned off upon the return from the subroutine. Thus the tracing may inadvertently extend beyond an intended range.

After a program is prepared with MSGLVL subroutine calls in it, it should be executed with a "/TRACE" switch, i.e., a command of:

```
.EXECUTE   FLNAME.FOR/TRACE
```

Later, after debugging is completed and program is corrected. The MSGLVL call statements can be left in by recompiling without the TRACE switch. (Do not forget to delete the old REL file first.)

Example: The NEWTON subroutine is revised to become the following:

```
      SUBROUTINE NEWTON(A,B,C,D,E,ITER,NPLACE,ROOT)
      DIMENSION X(2)
      CALL MSGLVL(3)  !***Turn on Tracer
1     X(1)=-B/A
2     ERROR=10.**(-NPLACE)
3     F1=FUNCP(A,B,C,D,E,X(1))
4     IF(F1.EQ.0.)X(1)=2.*X(1)
      CALL MSGLVL(1)  !***Turn off tracer
      DO 10 I=1,ITER
      X(2)=X(1)-FUNC(A,B,C,D,E,X(10))/FUNCP(A,B,C,D,X(1))
      IF(ABS((X(2)-X(1))/X(1)).LE.ERROR)GOTO 20
      X(1)=X(2)
10    CONTINUE
      WRITE(6,100)ITER
100   FORMAT(/'*** NO CONVERGENCE WITHIN',I5,' ITERATIONS.')
      RETURN
20      ROOT=X(2)
      RETURN
      END
```

Other program units, such as the main program, the function FUNC and FUNCP remain the same as those in Section 4.11. Upon execution by the command of "EXECUTE TEST/TRACE", the following printout was obtained:

```
<1><2><3>
Call to routine FUNCP from routine NEWTON
Return to routine NEWTON from line 5 of routine FUNCP
<4>
  etc.
```

The statement labels are enclosed between angle brackets: <2> means Statement labeled "2" (not Line 2), and <2>*6 means Statement labeled "2" 6 times. The advantage of using this routine over TRACE is that the compiler listing is not needed to determine the line numbers in a program.

ON-LINE DEBUGGING BY AN INTERACTIVE PROCESSOR

4.16 The FORDDT Processor

FORDDT is an interactive processor used to debug a FORTRAN program by controlling its execution. Using the symbols created by the compiler, FORDDT has the following capabilities:

(1) To examine and modify the data;
(2) To examine, specify or modify a FORMAT;
(3) To set breakpoints;
(4) To trace the source program statement by statement.

Before calling the FORDDT debugger, you must prepare REL files that are compiled with the compiler debugging facilities, such as tracing, accommodations for breakpoints, dimension checks, etc. This must be done even if there are already compiled REL files for the programs (but without debugging features). This is done by a monitor command:

```
.COMPILE  /COMPILE/DEBUG:ALL  list
```

where "/COMPILE" and "/DEBUG" are global switches placed in front of the programs "list". This will force a new compiling with debugging facilities loaded. However, do not force a compiling to those programs that don't belong to you, such as the system library, IMSL, etc. Afterward, FORDDT may be called by:

```
.DEBUG  list, SYS:FORDDT
```

where *list* is a complete list of files that are needed for execution. FORDDT will respond with:

```
STARTING FORTRAN DDT
>>
```

where ">>" is a prompting signal that FORDDT is ready to accept FORDDT commands. The details for the FORDDT may be found in Appendix E of Reference 9. A basic set of the FORDDT commands is included here with explanation and examples.

Certain prelimianry FORDDT rules are first explained:

(1) A FORDDT command consists of a keyword and optional parameter(s).

(2) Program data may be accessed by refering to their FORTRAN variable names.

(3) General array specifications is NAME(S1,S2,...,Sk). Variation of this specification are:

NAME	the entire NAME array
NAME(7)	the 7th element in the NAME array
NAME(K)	A subscripted element
NAME(3)-NAME(10)	Elements in a specified range

(4) Standard numeric convention applies.

(5) Statement number is represented by an unsigned integer, e.g., 100. The line number must be preceded by a pound sign(#), e.g., #100.

(6) FORDDT can specify numerical mode for data input/output by giving the "mode identifier." The mode identifier codes are defined as follows:

Code	Meaning	Example
A	ASCII(left-justified)	/LEFT/
C	Complex	(1.234,-6.543)
D	Double Precision	123.4567890
F	Real	1.2345
I	Integer	1234
O	Octal	7777
R	RASCII(right-justified)	\RIGHT\

4.17 Basic FORDDT Commands

Of the eighteen commands available in the FORDDT, a subset of eleven frequently used commands will be presented below:

Keyword	Parameter	Explanation
OPEN	name	To open the named unit (subprogram) and allow all variables within that unit to be accessible to FORDDT. If *name* is omitted, it means to re-open the main program. When FORDDT is called, the main program is automatically opened. Later, when one open command is applied after a preceding one, the unit previously opened is automatically closed. Therefore, at any given time, not more than one program unit may be opened.

Example	Function
.DEBUG TEST.FOR, SYS:FORDDT	
>>(command)	Main program opened for debugging
>>...	
>>OPEN SUB1	Open subprogram SUB1; close main program.
>>OPEN	Open main program again; close SUB1

START		To start your program at the main program entry point.
MODE	list	To define the "mode identifiers", or the display formats, for succeeding TYPE commands of FORDDT, and *list* contains one or more of the mode identifier codes separated by commas. The mode identifier codes are those defined in Section 4.16. The default mode is the floating point format, and output will return to the default mode with a "MODE" command with no argument.

Example	Function
>>MODE F,F,I	Set MODE for the next TYPE command.
>>TYPE A,B,K	Type A,B,K per MODE defined before.
>>MODE	Return to all floating point format.

TYPE list

To type out values of variables listed in the format defined by the last MODE command. "Print modifier" may be used to alter the format temporarily just for the current TYPE command. A print-modifier has a format of "/code" where the "code" is a mode identifier. If a print-modifier is placed after a variable name, only that variable output format is altered temporarily. If a print-modifier is placed before a variable name, all variables in that TYPE command after the modifier are temporarily set except those with individual modifiers. See examples below:

Example	Function
>>MODE	Reset MODE to default floating point.
>>TYPE A,B,C	Type A,B,C in floating point.
>>TYPE A,B,C/I	A,B as real, C in integer format
>>TYPE A,/I B,C	A as real, B and C as integer format
>>TYPE /I A,B,C	All in integer format
>>TYPE /I A,B,C/F,D/O,E/A,K(1)-K(10)	A & B in integer, C as floating point, D in octal, and E in ASCII format, K(1) through K(10) inclusive in integer format

ACCEPT name/mode value

To modify the values of listed variable names, where:

name = the name of the variable, array, array element, or array element range to be modified. If an array name is given without a subscript, the entire array will be modified.

mode = format of modifying data. Use mode identifier code for MODE.

value = new data for the variable

Example: >>ACCEPT A 1.23
Function: Set A=1.23 (default MODE=floating point)

Example: >>ACCEPT B/C (1.2,0.3)
Function: Set B=(1.2,0.3) --- B set in complex mode

Example: >>ACCEPT X(2)-X(9) 0.0
Function: Set X(2) through X(9) to 0.

Example: >>ACCEPT X 0.0
Function: Set the X-array to 0.

Example: >>ACCEPT FLNAME/A/LONG 'SAMPLE.DAT'
Function: Set FLNAME='SAMPLE.DAT' (2 words)

Example: >>ACCEPT 10 (1X,8F8.2)
Function: Set the FORMAT labelled as Statement 10 to the given new form. Work only for changing an old FORMAT to another with equal or shorter field.

GROUP n list — To set up a string of text for input to a TYPE command, where:
n = group number, 1 through 8.
list = a string of TYPE statemnts to be called.

Example: >>GROUP 1 A,B,C,/I I,J,K
Function: Store "A,B,C,/I I,J,K" as Group 1 data. Future output FORDDT commands for this group of data may be simplified into "TYPE /1"

Example: >>GROUP 1
Function: List Group 1 data names.

Example: >>GROUP
Function: List all stored GROUP lists.

PAUSE p — To set a breakpoint at point "p", where "p" is any label, line number, or sunroutine entry in your opened program unit. A maximum of ten breakpoints may be set at one time. Each PAU command can set only one breakpoint. When a pause is encountered, execution is suspended at that point and control is transferred to FORDDT. At that point, examination or modification of data can be made.

Example	Function
>>PAU 50	Set breakpoint at statement No. 50.
>>PAU #50	Set breakpoint at Line No. 50.
>>PAU #50 TYPING /2	Set breakpoint at Line#50. When paused there, type out data group 2. At least one blank is required between "TYPING" and "/2".

CONTINUE n — To ask the program to resume execution after a FORDDT pause, and the program will run until the nth occurrence of the given pause or until a different pause is encountered. The default n is 1.

REMOVE p — To remove the pause at "p" from the program set up by a previous pause command. If "p" is omitted, it will remove all pauses set up.

WHAT — To display on the terminal the name of the currently open program unit and any currently active pause settings.

STOP — To terminate the program execution, close all files, and return to the monitor.

4.18 A FORDDT Example

The example started in Section 4.11 will again be used to illustrate the FORDDT usage. The program listing by the program unit names is given below. Those statements that will be used as breakpoints are marked with check marks "✔".

Program Unit MAIN.

```
00001            CALL NEWTON(1.,-11.,9.,8.,20.,50,3,ROOT)
00002            TYPE 10, ROOT
00003    10      FORMAT(' ROOT = ', F8.2)
00004            END
```

Program Unit NEWTON:

```
00001    *
00002          ✔SUBROUTINE NEWTON(A,B,C,D,E,ITER,NPLACE,ROOT)
00003           DIMENSION X(2)
00004           X(1)=-B/A
00005           ERROR=10.**(-NPLACE)
00006          ✔F1=FUNCP(A,B,C,D,E,X(1))
00007           IF(F1.EQ.0.)X(1)=2.*X(1)
00008           DO 10 I=1,ITER
00009           X(2)=X(1)-FUNC(A,B,C,D,E,X(10))/FUNCP(A,B,C,D,X(1))
00010           IF(ABS((X(2)-X(1))/X(1)).LE.ERROR)GOTO 20
00011           X(1)=X(2)
00012    10    ✔CONTINUE
00013           WRITE(6,100)ITER
00014    100    FORMAT(/'*** NO CONVERGENCE WITHIN',I5,' ITERATIONS.')
00015           RETURN
00016    20     ROOT=X(2)
00017           RETURN
00018           END
```

To check whether correct parameters were passed.

To check parameters before passing on to FUNCP

To check condition at the end of each loop

Program Unit FUNC:

```
00001    *
00002          ✔FUNCTION FUNC(A,B,C,D,E,X)
00003           FUNC=A*X**4+B*X**3+C*X**2+D*X+E
00004           RETURN
00005           END
```

To check passed parameters

Program Unit FUNCP:

```
00001    *
00002          ✔FUNCTION FUNCP(A,B,C,D,X)
00003           FUNCP=4.*A*X**3+3.*B*X**2+2.*C*X+D
00004           RETURN
00005           END
```

To check passed parameter values

The placement of breakpoints is important. If a statement or a line is designated as a breakpoint by the PAUSE command, the program execution will pause before the execution of that line or statement. The breakpoints should be placed strategically at the following places:

(1) Subprogram entry points to check if parameters are being successfully passed;

(2) Just before returning to the calling programto see what kind of values are being passed back;

(3) After an input step to see if input operation is successful;

(4) Just before outputing a value if error report indicating a format overflow;

(5) A sampling point in the DO-loop to see how iterations are going;

(6) Place after a division to check if there is any divide-by-zero erorr;

(7) Places you feel to be critical or suspect where errors are being generated.

If a breakpoint is placed inside a DO-loop, be wary of large amount of FORDDT outputs. This can be circumvented by applying a FORDDT command "CONTINUE n" when ready to continue onto that breakpoint. This will allow the execution to repeat "ignore that breakpoint" for n times.

The remainder of this section is a reproduction of the terminal printout with annotations. The user's typings are in italics, and suitable comments are in handwritings.

```
.DEBUG  TEST.FOR/DEBUG:ALL, SYS:FORDDT
FORTRAN 5A(621): TEST FOR
MAIN    OCTAL PROG SIZE=56
NEWTON  OCTAL PROG SIZE=420
FUNC    OCTAL PROG SIZE=52
FUNCP   OCTAL PROG SIZE=56
LINK:   Loading
[LNKDEB DDT execution]

STARTING FORTRAN DDT

>>OPEN NEWTON
>>PAUSE #2
>>PAUSE #6 TYPING /1
>>GROUP 1 A,B,C,D,E,X
>>PAUSE 10 TYPING /2
>>GROUP 2 X
>>OPEN FUNC
>>PAU #2
>>OPEN FUNCP
>>PAU #2
>>START

PAUSE AT ROUTINE NEWTON
ARGUMENTS ARE:
        =   1.000000  (A
        =  -11.00000  (B
        =   9.000000  (C
        =   8.000000  (D
        =   20.00000  (E
```

Call for FORDDT to debug TEST.FOR
Make sure you have already deleted the old REL file of TEST.

Open the unit NEWTON. Set breakpoints at Line #2, Line #6, and Statement #10.
Define variable groups 1 & 2 as shown.
Open the unit FUNC
Open the unit FUNCP
Begin execution and debug

First pause at entrance point of Newton.
Parameter values are printed out automatically in the order of arguments.
No variable name is given for these data

```
        =                50   (ITER)
        =                 3   (NPLACE)
        =  0.0000000E+00      (ROOT)

[IMPLICIT OPEN NEWTON]

>>CONTINUE

PAUSE AT L#6 IN NEWTON
A       =    1.000000
B       =   -11.00000
C       =    9.000000
D       =    8.000000
E       =    20.00000
X(1)    =    11.00000
X(2)    =  0.0000000E+00

>>CONTINUE

PAUSEW AT ROUTINE FUNCP
ARGUMENTS ARE:
        =    1.000000      A
        =   -11.00000      B
        =    9.000000      C
        =    8.000000      D
        =    20.00000      X
        =    11.00000      ?

[IMPLICIT OPEN FUNCP]

>>CONTINUE

%FRSSRE Subscript range error on line 9 of NEWTON
                Subscript 1 of array X = 10

NEWTON[7037] called from MAIN.+5[6732] with 8 args of type F,F,F,F,F,I,I,F

PAUSE AT ROUTINE FUNC
ARGUMENTS ARE:
        =    1.000000      A
        =   -11.00000      B
        =    9.000000      C
        =    8.000000      D
        =    20.00000      E
        = -0.7939503E+32   X(1)

[IMPLICIT OPEN FUNC]

>>CONTINUE

%FRSAPR Floating overflow at FUNC+17[7440]
FUNC[7421] called from NEWTON+12[7151] ith 6 args of type F,F,F,F,F,F
NEWTON[7037] called from MAIN.+5[6732] with 8 args of type F,F,F,F,F,I,I,F

(more of similar message)

PAUSE AT 10 IN NEWTON
[IMPLICIT OPEN NEWTON]

X(1)    =  -0.1106965E+36
```

Machine will pause here to allow inspection of data.

Continue execution until next breakpoint

Next pause at line #6. Automatic output of Group 1 data defined.

These are data before calling FUNCP

Next pause at entry point of FUNCP.

These are data passed into FUNCP

Dimension out-of-bound error found

Next pause at entry point of FUNC.

Note the magnitude of X(1)

Error messages from FORDTS

Next pause at Statement Label 10, or at the end of first loop.

```
X(2)    =  -0.1106965E+36
>>STOP
End of execution    FOROTS 5B(1001)
CPU time: 0.65  Elapsed time: 1:41.67

No. of          Error
Errors          Type
9               Floating overflow

EXIT
```

Senseless to go any further

After inspecting the debugging printouts, two errors in the subprogram NEWTON were found:

(1) The argument "E" in line #6 should not be there;

(2) In line #9, X(10) should be X(1).

After errors are identified, UPDATE editor is used to correct the error, and the program is run again. The terminal printout is shown below:

```
.UPDATE TEST.FOR
      CALL NEWTON(1.,-11.,9.,8.,20.,50.,3,ROOT)
>$AT/FUNCP(A,B,C,D,E/; CHANGE/,E//
      F1=FUNCP(A,B,C,D,X(1))
>$AT/X(10)/; CHANGE/X(10)/X(1)/
      X(2)=X(1)-FUNC(A,B,C,D;E,X(1))/FUNCP(A,B,C,D,X(1))
>$END
2 blocks written on TEST.FOR[115103,320571]

.EXECUTE TEST.FOR
FORTRAN 5A(621): TEST.FOR
MAIN.   OCTAL PROG SIZE=53
NEWTON   CTAL PROG SIZE=210
FUNC    OCTAL PROG SIZE=52
FUNCP   OCTAL PROG DUZE=55
LINK:   Loading
[LNKXCT TEST execution]

ROOT =    10.0

End of execution    FOROTS 5B(1001)
CPU time: 0.02  Elapsed time: 0.02
EXIT
```

Thus, the laborious effort of debugging finally pays off.

MAY YOU CATCH ALL YOUR BUGS.

EXERCISE

The instructor should select a programming exercise of moderate difficulty for a problem for which the algorithm is well-understood by the class. Follow the procedure outlined below to the completion of a successful programming session:

(1) Set up a flow chart specification of the problem. Check if it is correct.

(2) Code the program according to the flow chart.

(3) Generate a flow chart based on the code and using the program FORFLO. Compare the generated flow chart with the problem specification. If there is any discrepancy, correct the code.

(4) Compile the program. From the compiler error list, correct all syntax errors and transcription errors not caught before.

(5) Organize a code-inspection and walkthrough session for a critique, and correct the errors identified.

(6) Run the program, using a test data set. If the run is not successful, debug on-line by any or combination of the on-line debugging techniques and aids covered in this chapter.

REFERENCES

1. PROGRAM DEBUGGING, A. R. Brown and W. A. Sampson, McDonald & Company, Ltd., London, Great Britain; 1973.

2. SOFTWARE DEBUGGING FOR MICROCOMPUTERS, Robert C. Bruce, Reston Publishing Company, Reston, Virginia; 1980.

3. THE ART OF SOFTWARE TESTING, Glenford J. Myers, John Wiley & Sons, Inc., New York; 1979.

4. FUNDAMENTALS OF FLOWCHARTING, T. J. Schriber, John Wiley & Sons, Inc., New York; 1969.

5. FLOWCHARTING, M. . Farina, Prentice Hall, Inc., Englewood Cliffs, N. J.; 1970.

6. PROGRAM STYLE, DESIGN, EFFICIENCY, DEBUGGING AND TESTING, by Daniel Van Tassel, Prentice-Hall, Inc., Englewood Cliffs, NJ; 1974.

7. FORTRAN PROVERBS FOR FORTRAN PROGRAMMERS, by Henry F. Ledgard, Hayden Book Compnay, Inc., Rochelle Park, NJ; 1975.

8. FORTRAN-10 USERS GUIDE, DEC-10 Documentation-2, Computer Center, University of Pittsburgh, Pittsburgh, Pennsylvania; 1977.

9. FORTRAN PROGRAMMER'S REFERENCE MANUAL, No.AA-0944E-TB, Appendix B, Appendix E and Appendix F, Digital Equipment Corporation, Maynard, Massachusettes; 1977.

CHAPTER 5

MODELING AND SIMULATION BY CSMP

Modeling and simulation are important tasks in science and engineering. Instead of the event-type or discrete-type modeling and simulation, this chapter will deal with modeling and simulation of a continuous system. A high-order language simulation program will be presented: Continuous System Modeling Program (CSMP). This program was originally developed for the IBM/360, and has since found wide acceptance in its application and adapted to many other computers including the DEC System-10.*

INTRODUCTION

5.1 Dynamic Modeling of Systems

One of the most important tasks in the disciplines of applied sciences and technology, including engineering, is the study of an existing system by analyzing its characteristics or the study of a proposed system by its synthesis under a set of prescribed objectives. Under most circumstances, it is not possible or practical to isolate that system, to dismantle it, and to perform the study. Therefore, it is necessary to construct a model, and examine the model performance by subjecting it to varying internal and/or external conditions.

In all areas of applied sciences, whether they are the physical, life or social sciences, the tasks of modeling and simulation are important elements of their analytical studies. Although the fields of disciplines may vary widely, the task of modeling and simulation may be summarized by the block diagram shown in Figure 5.1.

Beginning with the laws of nature (physics, chemistry, biology, etc.) and the laws of society (sociology, economics, law, etc.), the behavior of a model under various conditions may be described with mathematical language. Such a description, under most cases, takes on the form of a set of mathematical equations. The equations may be Boolean logic, algebraic, transcendental,

---

*Adaptation for the Pitt installation by Michael A. Matzek, Computer Center, University of Pittsburgh.

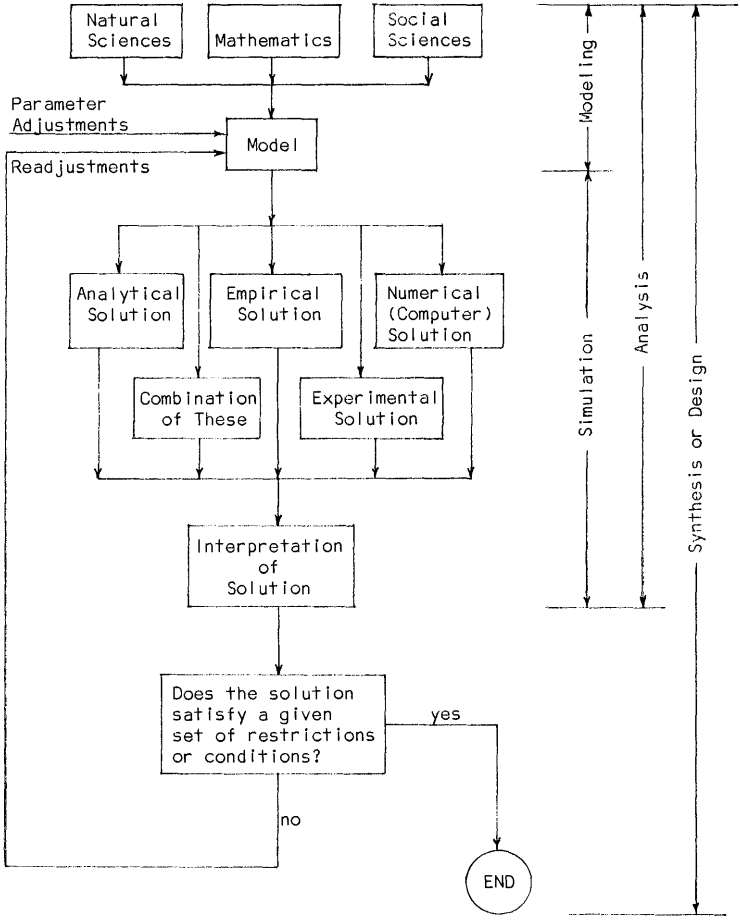


Figure 5.1 Modeling and Simulation in System Studies

differential, integral, difference, or combinations of these. The result is called a mathematical model.

A solution of the model can be obtained by many different ways. Applied mathematicians and analytically inclined scientists and engineers have devoted to find analytical solutions, where the solutions are preferably expressed in closed forms. Many engineering practitioners have found it convenient to develop, in addition to the analytical solutions, empirical ways of solution --- ways that have no rigorous analytical ground but they work. Other people find that to them the most effective way is by experiments such as in the areas of physics, chemistry, metallurgy, etc. With the advent of digital computers, another way becomes practical, that is, by digital computation. CSMP modeling and simulation falls into the last category.

Once the solution is obtained, its interpretation and conclusions are then drawn. Concluded at this point is a process of study called System Analysis.

When the system is analyzed with reference to a set of prescribed conditions or objectivess, one must further examine whether the result of the analysis satisfies these objectives. If not, adjustments of the model, either in its structure or in its parameter values or both, must be made. The process of solution is then repeated until the objectives are met. Concluded at this point is a process of system study called System Synthesis or System Design.

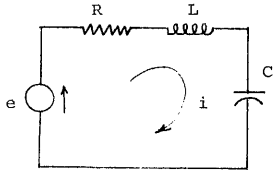
The mathematical description of many system processes encountered in engineering, scientific and societal problems is often made with respect to time as an independent variable. In such a description, the parameters, their changes and their rates of changes, etc., can be interpreted by applying the basic laws of natural sciences and social sciences. Resultant models will describe the time behaviors of the systems, and hence they are called dynamic models.

In most cases, the time behavior of a system may be described by an equation containing differential quantities, i.e., a differential equation. Consequently, facility in formulating differential equations and knowledge of how to obtain their solutions become the most important skills required in the system studies.

5.2 Differential Equations

When a differential equation is formulated from a real system, more often than not, it cannot be solved unless simplifying assumptions are made. After these assumptions are made, the result may be obtained by rigorous mathematical methods. An alternative is to approximate the differential equation as a difference equation, which may then be solved by performing a numerical integration. This is where a computer comes in. Although the result is only an approximation to the true solution of that equation, the error can be controlled and kept within a prescribed bound. Shown below are two examples of how the differetial equation model for a system may be formulated.

Example: Consider the following RLC circuit.



The voltage across each element is:

$$v_R = R\,i \qquad \text{(Ohm's Law)} \qquad (1)$$

$$v_L = L\,\frac{di}{dt} \qquad \text{(Lenz's Law)} \qquad (2)$$

$$v_C = \frac{1}{C}\int i\,dt \qquad \text{(Faraday's Law)} \qquad (3)$$

Then, by Kirchoff's Law, we write:

$$v_L + v_R + v_C = e \qquad (4)$$

$$L\,\frac{di}{dt} + Ri + \frac{1}{C}\int i\,dt = e \qquad (5)$$

Differentiating both sides, and letting de/dt=f(t):

$$L\,\frac{d^2i}{dt^2} + R\,\frac{di}{dt} + \frac{1}{C}\,i = f(t) \qquad (6)$$

The differential equation becomes a circuit model, the solution of which will give the time-behavior of the circuit.

The same circuit may be formulated into a model using the state variable approach. Suppose we choose "i" and "v " as the state variables. Then the equations (3) and (5) may be rewritten as:

$$\frac{dv_C}{dt} = \frac{1}{C}\,i \qquad (7)$$

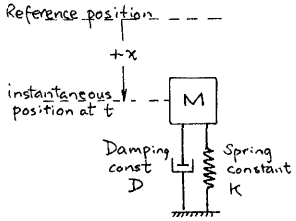
$$\frac{di}{dt} = \frac{e}{L} - \frac{R}{L}\,i - \frac{v_C}{L} \qquad (8)$$

These two simultaneous differential equations may be organized into a matrix form:

$$\begin{bmatrix} \frac{dv_C}{dt} \\ \\ \frac{di}{dt} \end{bmatrix} = \begin{bmatrix} 0 & \frac{1}{C} \\ \\ -\frac{1}{L} & -\frac{R}{L} \end{bmatrix} \cdot \begin{bmatrix} v_C \\ \\ i \end{bmatrix} + \begin{bmatrix} 0 \\ \\ \frac{e}{L} \end{bmatrix} \qquad (9)$$

Such a matrix differential equation is then referred to as a state-variable model of the circuit.

Example: The following diagram represents a simplified version of a Pogo stick, or a landing gear of an aircraft.



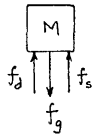
Consider the mass as a free body, and analyze the forces that act on it:

Gravitational: $f_g = Mg$ (10)

Spring force: $f_s = Kx$ (11)

Damping force: $f_d = D \frac{dx}{dt}$ (12)

Free Body Diagram



Net force in the downward direction = $f_g - f_s - f_d$

Thus, by Newton's Second Law:

$$M \frac{d^2x}{dt^2} = Mg - Kx - D \frac{dx}{dt} \quad (13)$$

or,

$$M \frac{d^2x}{dt^2} + D \frac{dx}{dt} + K x = M g \quad (14)$$

This is a mathematical model for the given mechanical system. Note the mathematical similarity between the models in the two examples given here.

5.3 Preparation for Digital Computer Solution

First, consider a first-order differential equation:

$$\frac{dx}{dt} + f_1(x,t)\ t = f_2(x,t) \quad (15)$$

where x is the dependent variable and t is the independent variable. Equation (15) may be reduced to a general form of:

$$\frac{dx}{dt} = f_2(x,t) - t\ f_1(x,t) = f(x,t) \quad (16)$$

Hence,

$$x = \int f(x,t)\ dt \quad (17)$$

If the initial condition is known (at t=0, x=x0), then equation (17) may be rewritten as:

$$x = x_0 + \int_0^t f(x, t)\, dt \tag{18}$$

Therefore, the solution of a first-order differential equation involves basically a numerical evaluation of an integral. Thus, depending on the method of numerical integration used, different methods of solution of differential equation may be derived. The users of CSMP will have options of choosing any one of the following methods of numerical integration:

Fixed-step methods:
- Rectangular integration
- Trapezoidal integration
- Simpson's rule integration
- Adams (second order) integration
- 4th order Runge-Kutta method, fixed interval

Variable-step methods:
- 4th order Runge-Kutta method, variable step
- 5th order Milne method

The details of these methods are available in any standard numerical methods text. It is sufficient here to say that the numerical solution of a first-order differential equation is a highly developed and important field in numerical analysis.

For the high-order equations, the general approach of their numerical solutions is to reduce each high-order equation to a set of simultaneous first-order equations. This can be illustrated by an example:

Example: Let us again consider that example of a mechanical system:

$$M \frac{d^2x}{dt^2} + D \frac{dx}{dt} + K x = M g \tag{14}$$

or,

$$\frac{d^2x}{dt^2} = g - \frac{D}{M} \frac{dx}{dt} - \frac{K}{M} x \tag{19}$$

First, define a new set of dependent variables:

$$x_1 = x \qquad \text{and} \qquad x_2 = \frac{dx}{dt} \quad \left(\text{or} = \frac{dx_1}{dt}\right)$$

Thus the equation (19) is now changed to two equations (20-21):

$$\frac{dx_1}{dt} = x_2 \tag{20}$$

$$\frac{dx_2}{dt} = g - \frac{D}{M} x_2 - \frac{K}{M} x_1 \tag{21}$$

The original second-order equation is now changed to a set of two first-order equations. Therefore, the first-order equation method of solution can now be applied.

This process of reducing a high-order differential equation to a set of first-order equations may be expanded to a general nth order equation.

Let the nth order equation be expressed as:

$$x^{(n)} = f(x, t, x', x'', x''', \ldots, x^{(n-1)}) \tag{22}$$

Note that the superscripts are orders of derivatives, not the powers. In the process of reducing a set of (n) first-order equations, the first (n-1) equations are simply new definitions of the derivatives.

Define: $x = x_1$

$$\left.\begin{aligned} \frac{dx_1}{dt} &= x_2 \qquad (or,\ x_2 = x') \\ \frac{dx_2}{dt} &= x_3 \qquad (or,\ x_3 = x'') \\ &\cdots \\ \frac{dx_{(n-1)}}{dt} &= x_n \qquad (or,\ x_n = x^{(n-1)}) \end{aligned}\right\} \tag{23}$$

and Equation (22) becomes:

$$\frac{dx_n}{dt} = f(x_1, t, x_2, x_3, \ldots, x_n)$$

Students of Circuit Theory, System Theory, Automatic Control System, etc., will recognize that this is exactly the same as the process of formulating the state-variable equations. Equations (23) now consist of a set of first-order equations and the first-order equation methods can now be applied.

Integration of the set of equations (23) yields:

$$\left.\begin{aligned} x_1 &= \int x_2 dt = x_{1_o} + \int_0^t x_2 dt \\ &\cdots \\ x_{n-1} &= \int x_n\, dt = x_{n-1_o} + \int_0^t x_n dt \\ x_n &= \int f(x_1, t, x_2 - x_3, \ldots, x_n) dt = x_{n_o} + \int_0^t f\, dt \end{aligned}\right\} \tag{24}$$

where x_i are the initial coditions of x_i at t=0. Thus for each equation, one step of numerical integration is taken to get a new point of xi. When all of equations (24) have taken that single step, the sequence is repeated for the next increment.

If the process of numerical integration can be written as a subprogram, it would require two parameters: the initial condition and the integrand function. Suppose such a subprogram is available and defined as:

```
Y = INTGRL(IC,X)
```

where IC = initial condition of Y, and X = integrand expression. Then the set of equations (24) may be written as:

```
X1 = INTGRL(X10, X2)
X2 = INTGRL(X20, X3)
                                    } (25)
     ...

XN = INTGRL(XNM1, F)
```

The INTGRL is a CSMP library function. The high-order differential equations may be represented simply as a series of INTGRL function in CSMP language.

5.4 CSMP as a High-Order Language (HOL)

While the digital computer is a tremendously powerful tool in the studies of systems, the tasks of programming can be prohibitively tedious and expensive. In the design of programming languages, a set of hierarchical structure is established. In the increasing order of hierarchical structure, they are:

1. Machine language
2. Assembly language, such as MACRO-10
3. Compiler language, such as FORTRAN and COBOL
4. High-order language, such as CSMP, ECAP, CORNAP, etc.

In this hierarchy, the programming efforts in each higher level will be considerably reduced, but the rigidity of programming is also considerably increased. Thus we are trading off machine utilization in favor of human resource utilization. In the computer system software development, the machine utilization has overriding importance, so we would favor languages of a lower hierarchy level, such as the assembly language. Thus, the basic system softwares, such as the operating system, various language compilers, and utility routines, are written in assembly language.

However, for applications where human resource utilization may be more important than an efficient machine utilization, engineers and scientists will use a compiler language or a high-order language to solve their problems.

But FORTRAN programming can be very complicated too. In the numerical solution of differential equations, one could use FORTRAN to program the solution. But the effort would be monumental if we included many options, control of accuracy, selection of output formats, etc.

High-order languages are therefore designed to ease such problems in the area of special applications. CSMP (Continuous System Modeling Program) is one such language.

CSMP was originally developed for the IBM System/360, but has since found wide acceptance. Modifications of the IBM versions became available for adaptation to other machines, including the DEC System-10. It is an application-oriented language that allows a problem to be prepared directly from either a block-diagram representation of the system or a set of ordinary differential equations. The language includes a basic set of functional blocks with which the components of a continuous system may be represented, and accepts application-oriented statements for defining the connections between these functional blocks. CSMP also accepts most FORTRAN statements, allowing the user to readily handle nonlinear and time-variant problems of considerable complexity. Both tabular and graphic output formats are available.

A typical CSMP program contains both CSMP statements and FORTRAN statements; it will also contain both CSMP functions and the conventional FORTRAN functions. After the CSMP program is prepared, it is first translated entirely to a FORTRAN program. This in turn is compiled, loaded, and executed with all the called CSMP functions and FORTRAN functions from the library. However, all these steps of translation, compiling, loading and execution are "transparent" to the user. To him, a single step of submitting a CSMP program to the CSMP processor is all that is required. That is the major power of a high-order language.

A CSMP PRIMER

5.5 Symbols, Constants, Operators, Functions and Labels

(1) Symbols CSMP symbols follow the same FORTRAN rules with the following exceptions:

A. All variables are real variables (even those beginning with I,J,K,etc.). Integer variables require a special declaration using the "FIXED" statement of CSMP.

B. Subscripted variables may not begin with the letter I, J, K, L, M, or N.

C. There is a group of CSMP-reserved names that should not be used by the user in a CSMP program. See page 66 of Reference 1.

(2) Constants All numeric constants are considered to be real, regardless of whether a decimal point is used. Only integer variables have integer constants. Other than this point, the standard rules of FORTRAN numeric constants apply. Double precision constants are allowed in CSMP III, but not in CSMP II.

(3) Operators The same standard rules of FORTRAN operators apply to CSMP, such as those for the operators + ,- , *, /, **, =, and the use of parentheses.

(4) Functions In addition to the standard FORTRAN library functions, such as SIN(X) and ALOG(X), there is a group of 34 CSMP functions. This is one of the most powerful features of the CSMP program. Selected functions will be discussed in a later section.

(5) Labels The first word of a CSMP data or control statement is a label word that identifies the purpose of the statement. These labels will be explained later.

5.6 Format of CSMP

The following are those format rules of CSMP statements that are different from those of standard FORTRAN:

(1) A CSMP statement may start anywhere within column 1 through 72. Statement materials outside column 72 will not be processed. A few specific label words must begin at column 1. These exceptions will be singled out as they occur.

(2) Continuation of a line is done by 3 consecutive periods (...) at the end of a line. A character in column 6 does not mean continuation in CSMP as in FORTRAN. A maximum of 8 continuation lines (9, including the first statement) are allowed.

Example:
```
XDDOT1=COEFF1*XDOT2+QUOt1*X2-COEFF*XDOT1- ...
       QUOT2*X2
```

Do not split a number or a variable name between a line and its continuation.

(3) A "*" character in column 1 means a comment line.

(4) Blank lines are allowed but ignored in the translation. It is usually used to make a program easier to read.

5.7 Structure of a CSMP Program

Basically, a CSMP program can be divided into 3 segments: the INITIAL segment, the DYNAMIC segment, and the TERMINAL segment.

(1) INITIAL segment This segment appears first in a CSMP program, and is used to define parameters, initial conditions, or calculations that need to be done only once. It is optional, and may not even be needed in a simple program. If it is used, there must be an INITIAL label line at the beginning of the segment.

Example:
```
INITIAL
PARAMETER R=50.0, L=1.25, C=0.25E-4
INCON I0=0.0
CONSTANT PI=3.14159
```

(2) DYNAMIC segment This is the heart of a CSMP program, and is a required segment. It contains the system model equations. If an INITIAL segment is not specified, the DYNAMIC segment is automatically incorporated, and no label is required. If an INITIAL segment is specified, the DYNAMIC segment must be headed with a line with a "DYNAMIC" label.

(3) TERMINAL segment This is the last segment of a CSMP program for computations performed after the simulation is finished. Or, it may be used to adjust parameters and reset conditions after one run in order to get ready for another run of the same problem.

5.8 SORT and NOSORT Sections

One of the most important features of the CSMP program is its sorting capability. Here, "sorting" does not mean the usual way of sorting by numbers or letters. Frequently, modeling of a system results in a system of simultaneous differential equations that must be processed in a particular order. In FORTRAN programming, the order is sequenced by the way in which the FORTRAN statements are arranged. In CSMP, we can delegate the responsibility of sequencing to CSMP by asking it to SORT. As a result, those calculations that will produce results needed in the later part of the iteration cycle will be done first. This then allows us to write the modeling equations without concern for their sequence. Therefore, the SORT capability of CSMP makes the solution of simultaneous differential equation by CSMP a virtual parallel operation.

On the other hand, there are computation that must be executed in the exact order specified, such as a FORTRAN conditional logic (IF) statement or a branching (GO TO) statement. They should be specified as NOSORT so that the given fixed order will not be disturbed.

The following types of statements should be placed in a NOSORT section:

(1) Conditional logic (the IF statement)

(2) Branching (any type of GOTO) statement

(3) Implicit arithmetic statement, such as X = X+1.0

(4) WRITE and FORMAT statements

The following SORT rules apply to the default conditions:

(1) INITIAL and DYNAMIC segments: If not labeled by NOSORT, the section is automatically regarded as SORT.

(2) TERMINAL segment: If not labeled by SORT, the section is automatically regarded as NOSORT.

CSMP STATEMENTS

There are three types of CSMP statements: Structure, Data, and Control Statements. A structure statement describes a functional relationship between the variables of the program. A collection of structure statements defines the system being simulated. A data statement assigns values to a variable. A control statement controls the operations and the quantities associated with the program, such as the step size, the print increment, the selection of methods, etc. These statement are explained below.

5.9 Structure Statements

A structure statement in CSMP is written in the form of:

VARIABLE = EXPRESSION

where the "EXPRESSION" follows the general rules of FORTRAN assignment statements. An expression may be simply a constant, another variable, a function, or a combination of these linked by arithmetic operators.

One special feature is the CSMP functions - they are the CSMP library functions available in addition to the standard FORTRAN functions. A complete list of CSMP functions are listed on pp. 9-16 of Reference 1. Tables 5.1A and 5.1B contain those more commonly used:

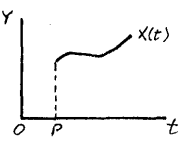
Name	CSMP Function Form	Function
Integrator	Y = INTGRL(IC,X)	$Y = IC + \int_{t_o}^{t} X\,dt$
	IC = initial condition; a constant or variable, but not an expresssion. X = integrand; a constant, a variable or an expression. Laplace form: $\frac{1}{s}$	
Derivative	Y = DERIV(IC,X)	Y = dX/dt
	IC = initial condition of derivative, or $\dot{X}(0)$ Laplace form: s	
Delay	Y = DELAY(N,P,X) N=number of sample points Laplace form: e^{-Ps}	 Y(t)=X(t-P), t≥P Y(t)=0 t<P
Real Pole	Y = REALPL(IC,P,X) IC = initial condition	Solution of: $P\frac{dY}{dt} + Y = X$
	Laplace form: $\frac{1}{Ps + 1}$	
Lead-Lag	Y = LEDLAG(P1,P2,X) Laplace form: $\frac{(P1)s + 1}{(P2)s + 1}$	Solution of: $P2\frac{dY}{dt} + Y = P1\frac{dX}{dt} + X$
Complex Poles	Y = CMPXPl(IC1,IC2,P1,P2,X)	Solution of: $\frac{d^2Y}{dt^2} + 2(P1)(P2)\frac{dY}{dt} + (P2)^2 Y = X$
	Laplace form: $1/(s^2 + 2(P1)*(P2)*s + P2^2)$	

Table 5.1A Selected CSMP Library Functions I

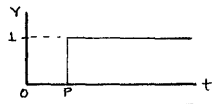
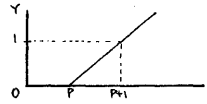
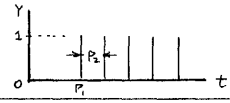
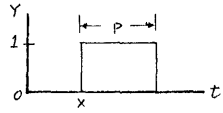
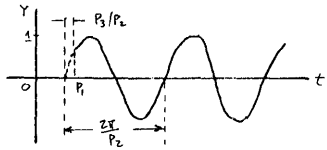
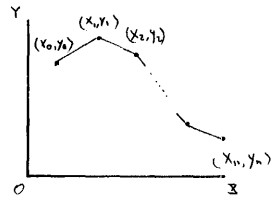
Name	CSMP Function Form	Function
STEP	Y = STEP(P) = unit step at P	
RAMP	Y = RAMP(P) = unit ramp at P	
IMPULSE	Y = IMPULS(P1,P2)	
PULSE	Y = PULSE(P,X) = unit height pulse triggered at time P	
SINE	Y = SINE(P1,P2,P3) P1=delay, seconds P2=angular frequency, radian/second P3=phase shift,radians	
Arbitrary Function Generation	Y = AFGEN(FUNC,X) (Linear interpolation between points) Y = NLFGEN(FUNC,X) (Quadratic interpolation between points) X = independent variable FUNC=function name	

Table 5.1B Selected CSMP Library Functions II

Much of the CSMP versatility is attributed to two major factors. One is the specially designed CSMP library functions and statements. The other is the SORT and NOSORT capabilities. The following are examples of how to write specific CSMP functions:

(1) INTGRL function

The INTGRL function, among all CSMP functions, is certainly one of the most frequently used functions. The format:

```
Y = INTGRL(IC,X)
```

represents a differential equation model of $\frac{dY}{dt} = X$, so that $Y = IC + \int_{t_o}^{t} X dt$

Example: Model the equation $\frac{dx}{dt} = v$, with initial condition x(0)=X0.

CSMP Statement:

```
X = INTGRL(X0,V)
```

Example: Model the simultaneous differential equations:

$$\frac{dv1}{dt} = v2 - v1$$

$$\frac{dv2}{dt} = 4\ v1 - t$$

Initial conditions: v1(0)=1.0
v2(0)=-2.0

The CSMP statements are:

```
V1 = INTGRL(1.0,V2-V1)

V2 = INTGRL(-2.0,4.0*V1-TIME)
```

Example: Although the INTGRL function follows the format of any standard FORTRAN library function, it has one restriction: The function must be placed at the right-most end of a statement.

```
Incorrect:    V1 = INTGRL(1.0,V2-V1) + 4.0*V2

  Correct:    V1 = 4.0*V2 + INTGRL(1.0,V2-V1)
```

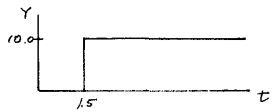
As a result, one CSMP statement may not contain more than one INTGRL function. They must be broken into several statements.

```
Incorrect:    FUNC = INTGRL(2.3,V1) + INTGRL(-0.9,V1-V2)

  Correct:    FUNC1 = INTGRL(2.3,V1)
              FUNC  = FUNC1 + INTGRL(-0.9,V1-V2)
```

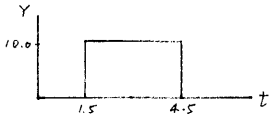
(2) STEP function

This represents a frequently used signal in finding the transient response of a system. The unit step function has an amplitude of 1. The amplitude of a desired step function may be set by a multiplier. The multiplier may be a constant, and in that case, the resulting function is a step function of amplitude specified. The multipler may be a time-varying function or expression, and in that case, the unit step function serves as a "switch" to turn on a time-varying function.

Examples:

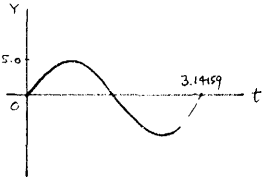



```
Y = 10.0 * STEP(1.5)
```



Single-shot multivibrator:

```
Y = 10.0*(STEP(1.5)-STEP(4.5))
```



Single-shot sinusoidal:

```
Y = 5.0*PULSE(2.0,0.0)*SINE(0.0,  ...
       2.0,0.0)
```

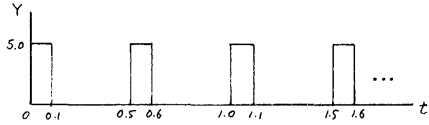
or,

```
Y = 5.0(STEP(0.0)-STEP(3.14159))...
      *SINE(0.0,2.0,0.0)
```

(3) IMPULS and PULSE functions

These two functions are often used together to produce a pulse train. See example below:

Example: Generation of a pulse train, 2 pulses per second, pulse width 0.1 second. Wave form is shown below:



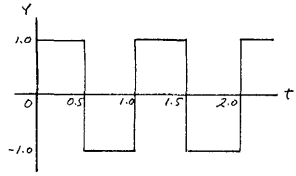

```
Y = PULSE (0.1, IMPULS(0.0, 0.5))
```

pulse width

spacing between pulses

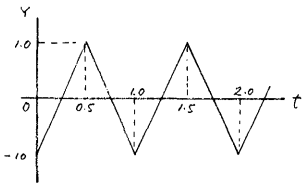
instant when the first pulse is triggered

Example:



A square wave generator

```
Y = 2.0*PULSE(0.5,IMPULS(0.0,1.0))-1.0
```



A triangular wave generator:

Let the pulse output of the last example be P. Then,

```
P = 2.0*PULSE(0.5,IMPULS(0.0,1.0))-1.0

Y = 4.0*(-1.0+INTGRL(0.0,Y))
```

(Question: Why 4.0?)

5.10 Data Statements

Data statements are those that assign numeric constants to variables in preparation to begin a CSMP run. Therefore, they are likely to appear in the INITIAL segment.

The format of a data statement begins with a label, followed by the assignments of numeric constants to certain variables.

INCON — To define variables that are used as initial conditions.
Example: INCON X0=2.5, V0=-4.56

PARAMETER — To define values of parameters that may be changed for different runs of simulation.
Example: PARAMETER VT=160.0, JK=1.5

Multiple parameter definitions may be written for multiple runs:
Example: PARAMETER VT=(160.,161.,162.,163.), JK=1.5
or, PARAMETER VT=(160.,3*1.), JK=1.5
These last two statements are equivalent. Note the repeat constant "3" must be written without a decimal point. Specified number of runs will be made for everything up to the statement of END. Maximum allowed: 50 runs, and one multiple-PARAMETER statement per sequence.

CONSTANT — To define values of variables that do not change values.
Example: CONSTANT PI=3.14159,HERTZ=60.0

FUNCTION — To define data points (in pairs of coordinates) of an arbitrary function. This is used in conjunction with either the function AFGEN or NLFGEN.
Example: FUNCTION FOT=(0.,0.),(.10,2.5),(2.0,-3.1),...
(3.0,5.6), (4.0,3.2)

```
Y = AFGEN(FOT,TIME)
```

CSMP treats the labels INCON, PARAMETER, CONSTANT the same way, and therefore they are interchangeable. However, separating them for their proper purposes would add to our clarity of program, and less likely to be misunderstood later.

5.11 Control Statements

These statements specify the translation, execution or output of the simulation. For example, one may use these statements to specify how large should be each increment in the iteration, in the print-plot diagram, or which method of solution should be used. In general, these statements may be mixed with the structure and data statements, and may appear in any order. Except for ENDJOB, COMMON and ENDDATA statements that must begin at column 1, the statements may start at any column. There are three kinds of control Statements:

(1) Translational control statements

INITIAL — A line with only the label INITIAL in it. This statement marks the beginning of the INITIAL segment.

DYNAMIC — This label marks the beginning of the DYNAMIC segment. This segment is terminated by a TERMINAL label.

TERMINAL — This label marks the end of the DYNAMIC segment and the beginning of the TERMINAL segment. The TERMINAL segment is terminated by the first END or CONTINUE statement.

END — The END statement marks the end of a simulation run. It will then allow more control statements to start a new run. If the END statement is followed by a STOP statement, then simulation runs are terminated. When a new simulation run is started by statements after the END statement, the independent variable TIME is reset to zero, and all initial conditions are reset.

CONTINUE — If a new run of simulation is to continue on the TIME-scale rather than reset it to zero, then a CONTINUE statement should be used instead of an END statement. Be careful to distinguish between a CONTINUE statement in CSMP and a CONTINUE statement in FORTRAN. Since a CONTINUE in FORTRAN is often used to terminate a DO-loop, it is always better to label a FORTRAN CONTINUE statement with a number.

STOP — An END followed by a STOP statement marks the termination of simulation and no new run is to be initiated.

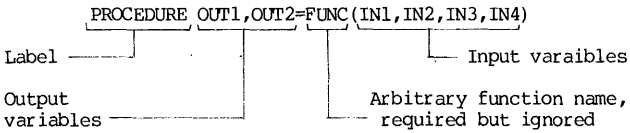
ENDJOB — This is the physical end of a CSMP package. It must begin at column 1.

RENAME — The standard name for the CSMP independent variable is TIME, and many reserved CSMP names are time-related names, such as DELT, DELMIN, FINTIM, PRDEL, and OUTDEL. Often, the independent variable for a simulation problem is not time. For example, in a beam deflection problem, the independent variable may be the linear displacement X. The names for the independent variable may be changed from TIME to X by "RENAME TIME = X". Then all reference to the independent variable TIME will be renamed to those with reference to X.

FIXED This label defines the listed variables on that line as integer variables. Without this statement, all CSMP variables are considered as real variables, including those beginning with a letter I, J, K, L, M or N.
Example: FIXED N,JOB

COMMON The statement contains just the label, beginning from column 1. This statement will make the values assigned for TIME, DELT, DELMIN, FINTIM, PRDEL and OUTDEL, and the values of those variables listed on PARAMETER, INCON, and CONSTANT lines available as a COMMON block available for any user-supplied FORTRAN subroutines.

PROCEDURE This statement leads off an entity of programs in the DYNAMIC segment that serves as a user-defined subprogram. Its format is given in the following example:



Listing the output variables is for the purpose of making them available later for PRINT or PRTPLOT statements. Variables not defined as output variables in a PROCEDURE statement may be used for later processing, but they will be unavailable for print or print-plot purposes.
The PROCEDURE statement is then followed by FORTRAN or CSMP statements, or their combinations.
The termination of a PROCEDURE is marked by a line with the label of ENDPRO.
The PROCEDURE section may be placed anywhere in the DYNAMIC segment, and SORT command will place the subprogram at the place in the order. But the internal sequence of the PROCEDURE subrpogram will not be sorted, and they will be executed in the exact order as they are given. As a result, full extent of FORTRAN and CSMP statements, including FORTRAN logic and branching statements, may be used.
Note: When FORTRAN and CSMP statements appear together in a PROCEDURE, the CSMP continuation rule (3 periods ...) applies for continuing a statement.

MACRO Line with the label MACRO will preceed a MACRO function, which has the following format:

The body of the MACRO for this example, and its termination line are:

```
X = RHO * COS(THETA)
Y = RNO * SIN(THETA)

ENDMAC
```

A MACRO must be placed at the beginning of the CSMP program ahead of any structure statement in the INITIAL or DYNAMIC segment. Once a MACRO function is so defined, it may be referenced an unlimited number of times in the program. The MACRO may contain FORTRAN or CSMP statements, but it should not contain any FORTRAN logical, branching, or I/O statements, nor should it contain any CSMP data and control statements.

SORT and NOSORT

In the processing of a CSMP program, calculations of many dependent variables are done with respect to the same independent variable, such as at the same instant of TIME. Such calculations are referred to as "parallel calculations". In any assignment statements, however, the righthand side cannot be processed until all the current values of the variables of the righthand side are known. This implies that these structural statements must be sequenced in a particular order. This ordering can be done automatically by the CSMP translator, if a label of SORT precedes the statements.

On the other hand, certain statements, such as the FORTRAN branching, logical decision and I/O statements must be executed in the exact order written. A NOSORT label preceding the statements will suppress the SORT algorithm for this group.

(2) Execution control statements

TIMER

The TIMER label is followed on the same line by various increment assignments. These increments have reserved names and default values:

Name	Meaning	Default Value
DELT	Increment for the independent variable.	DELT = $\frac{1}{16}$ MIN(PRDEL,OUTDEL) It is a good practice to always specify DELT.
DELMIN	Two methods, MILNE and RKS, use adjustable steps. Simulation will halt when: Adjusted DELT < DELMIN	
FINTIM	Max value of independent variable for the simulation run.	FINTIM=0 if it is not specified, and simulation will not even get started. Therefore, FINTIM should not be omitted.
PRDEL	Print output increment	PRDEL=OUTDEL if OUTDEL is given. PRDEL=FINTIM/100 if OUTDEL is not also given.

OUTDEL	Print-plot output increment	OUTDEL=PRDEL if PRDEL is given. OUTDEL=FINTIM/100 if PRDEL is not given also.

Example: TIMER DELT=0.1, PRDEL=0.5, FINTIM=10.0
Function: Specify in the simulation iteration that each increment of TIME should be 0.1 second. In the printout table, increment of output is).5. Simulation to be terminated when 10.0 seconds on the TIME scale is reached.

FINISH This is a label containing terminating logic conditions to supplement the FINTIM specification in the TIMER statement.

Example: TIMER FINTIM = 10.0
FINISH VEL = 1000.0, HEIGHT = YMAX
Function: During the simualtion run, the sign of (VEL-1000.0) and the sign of (HEIGHT-YMAX) is monitored at the end of each iteration. Simulation will be terminated when either of the three conditions is reached: reversal of signs of (FINTIM-10.), (VEL-1000.0), or (HEIGHT-YMAX).

RELERR
ABSERR These two labels respectively specify the relative and the absolute error bounds in the RKS and the MILNE methods.

Example: RELERR VEL = 1.0E-5
Function: This statement sets the relative error bound at .00001. When the relative error of VEL is lower than the bound, the step is adjusted, and the iteration takes steps of bigger size. These labels are only applicable to RKS and MILNE mehtods where iteration steps are adjustable.

METHOD This label followed by any one of the following seven words specifies the method of numerical integration. The default method is RKS.

ADAMS	Second-order Adams method
CENTRAL	A dummy routine that may be substituted by a user-supplied subroutine for integration.
MILNE	Variable-step, fifth-order, predictor-corrector MILNE method
RECT	Rectangular method of integration
RKS	Fourth-order Runge-Kutta method with fixed integration interval.
SIMP	Integration by Simpson's rule
TRAPZ	Integration by trapezoidal rule

Example: METHOD ADAMS

(3) Output Control Statements

In the following statements, the arguments for the labels are given in italics. The meaning of the arguments are:

list = a list of variables

string = a string of alphanumeric characters

label = a CSMP label

TITLE *string* — To print out the *string* as the title of each page of the print output. No continuation of line is allowed beyond one line.

LABEL *string* — To print out the *string* as the title of each page of the printplot output. No continuation of line allowed.

RANGE *list* — To print out the values of listed variables at minimum and maximum values of the independent variables.

PRINT *list* — To print (tabulated) listed variables versus the independent variable.

PRTPLOT *list*
PRTPLT — To produce plots by printer for the listed variables versus the independent variable.

The PRTPLOT statement is certainly the most frequently used output statement. There are a number of variations on how the listed variables may be given to vary the style of the plot. They are illustrated by examples below:

Example: PRTPLOT X,Y,Z
Function: To produce 3 printer-plots: X versus TIME, Y versus TIME, and Z versus TIME. (3 separate plots)

Example: PRTPLOT X(Y,Z)
Function: To produce 1 plot of X versus TIME, with values of Y and Z printed to the right of the plot as a table.

Example: PRTPLOT X(-1.0,1.0,Y,Z)
Function: Same as X(Y,Z) except with X-plot clipped at lower bound and upper bound of -1.0 and +1.0 respectively.

Example: PRTPLOT X(-1.0,,Y,Z)
Function: Same as X(Y,Z) except with the X-plot clipped at a lower bound of -1.0.

Example: PRTPLOT X(,1.0,Y,Z)
Function: Same as X(Y,Z) except with the X-plot clipped at an upper bound of +1.0.

PREPARE *list* — To prepare data for a X-Y plotter.

RESET *label* — To reset listed labels that control the increments of outputs, or increments of iterations.

RUNNING CSMP AT PITT

5.12 CSMP Job Preparation

CSMP is one of the high-order languages, in which the language primitives are at a high level of sophistication. In the execution of such a program, the general approach is to translate it successively down into a lower order language by translators, compilers, assemblers, and finally down to the machine language level for machine execution. Since FORTRAN compiler and assembler already exist, it is naturally expedient to design a high-level language that its associated translator would have the responsibility of only translating the high-level program to the level of FORTRAN. From that point on, the existing compiling-assembling mechanism can take over.

When a CSMP program is accepted by the CSMP processor, the processor first builds a FORTRAN subroutine named UPDATE.TMP and a data file in your disk. In generating the UPDATE.TMP* file, the processor accomplishes three major tasks:

(1) The statements in the SORT section are placed in the proper order.

(2) The proper transfer of control for the various segments and sections is established.

(3) COMMON statements are established to make the proper variables available between UPDATE and the CSMP modules.

After that, the FORTRAN compiler-loader takes over to build an execution file in the standard way, and control is then passed over to the main program or the calling program.

At the University of Pittsburgh, only CSMP II has been implemented on the DEC-10, and certain CSMP utilities are yet to be completed. To run a CSMP program on the Pitt DEC System-10, a minimum of 26K core is required, and more for larger programs. Therefore, CSMP programs can only be run as a batch job unless the user has a sufficently large time-sharing core allocation.

However, this does not mean that a CSMP job cannot be run on a terminal. As will be explained next, a CSMP batch job may be either submitted in cards at a card reader, or submitted throught a stored file from a terminal.

Thus, there are three common ways of running a CSMP program and they are:

(1) Card input with a CSMP deck,

(2) Card input a stored CSMP file,

(3) Terminal input with a stored CSMP file. Their preparations are outlined below:

---

*It is named with a TMP (for "temporary") extension so that it may be easily deleted later without affecting your other FORTRAN files in the disk.

(1) Card input, with a deck of CSMP program cards

```
$JOB card                 Prepare a card deck with a sequence order
$PASSWORD card            shown here.  Prepare the CSMP program also
$CSMP                     in cards.  Submit these cards in the usual
                          manner through a card reader.
  (CSMP deck)

$EOD
$EOJ
```

(2) Card input, with stored CSMP program

```
$JOB card                 Store a prepared CSMP program on disk, and
$PASSWORD card            name it, for example, as XYZ.CSM.
.R CSMP                   Prepare a card deck as shown, and submit
*XYZ.CSM                  the job at a card reader.
$EOJ
```

(3) To run a CSMP batch job at a terminal

Prepare two files and store them on disk. One is the CSMP program file and name it as, for example, XYZ.CSM. The other is a control file, and name it as, for example, ABC.CTL. The control file contains the following lines:

```
$JOB line ....      (See discussion below)
.R CSMP
*XYZ.CSM
$EOJ
```

If it is desirable to capture the output data for such purpose as editing, multiple copy printing, or preparing for plotter output, use the following control file:

```
$JOB line ....
.ASSIGN DSK 6
.R CSMP
*XYZ.CSM
$EOJ
```

After the execution of the program, the CSMP output will be captured as FOR06.DAT, and no printer output is produced, unless later the user applies the QUEUE command, such as:

```
.QUEUE FOR06.DAT/COPIES:3/FILE:FORT
```

The last switch /FILE is needed in order to handle the FORTRAN printer-control characters on the file.

Whether to submit a CSMP job from a card reader or from a terminal, the $JOB card should be specified in this manner:

(1) Make a request for core allocation for at least 26K. Larger program will require more.

(2) Make a request for CPU time allowance of 2 minutes. The standard allowance is 30 seconds.

(3) Specify the RJE station where the output is to be produced.

(4) Specify page limit large enough for the output.

Example:

```
$JOB EXER4[115027,320571]/CORE:26K/TIME:2:00/LOC:10/PAGES:50
```

5.13 CSMP Job Execution

For CSMP batch jobs in cards, just read in cards prepared as described in Section 5.12. Execution will take place when the batch job is executed, and output is produced in the usual way.

If the CSMP jobs are submitted at the terminal, use OPRSTK command to submit the job. (See also Chapter 7) Thus, at the terminal, issue a monitor command:

```
.OPRSTK name
```

as log as a control file has previously been set up and named as NAME.CTL.

If a user has a time-sharing core allocation of 26K or more, he can run the CSMP job on the time-sharing system by issuing a command at the terminal:

```
.R CSMP
```

When a prompt symbol "*" appears, give the name of the stored CSMP file. In this case, the output will be produced on the user's terminal, unless a monitor command of ".ASSIGN DSK 6" has been previously given. In the latter case, the output will be stored as FOR06.DAT.

5.14 Other Modeling and Simulation Languages

CSMP is a simulation language mainly for continuous system simulation by solving dynamic model formulated in terms of a set of ordinary or partial differential equations. There are other high-order languages that accomplish a similar purpose, and some of these languages are available for the DEC-10 machine. Presentations of these languages are outside the scope of this book. Therefore, they will only be listed as references:

(1) Analog-Computer Emulators These are languages emulating the simulations formulated for analog computers. They were once very popular in the 1960's when analog computers were still heavily depended on for continuous system simulation. The emulator MIDAS is one typical example. They are not in general use now.

(2) Simulation Language mainly for the solution of ordinary differential equations.

ASCL Initial-value problem solver, run on UNIVAC 1100 machine.

CSSL Continuous System Simulation Language, developed by the control

Data Corportion for CDC 6600 machine.

DAREP Differential Analyzer Replacement - Portable, University of Arizona. Coded in FORTRAN except a few machine dependent routines. Available for PDP-9, DEC-10, IBM/360, and CDC6600 machines.

EASY A dynamic analysis language that provides both modular modeling and modular simulation. Written in FORTRAN and run on CDC machines. Versions for DEC and IBM are being developed.

DYNAMO A language developed by the Industrial Dynamics Group at MIT. In spite of very crude method of differential equation solution (Euler's method), this language has been very popular among social scientists because of its simplicity. This language is available at Pitt.

MIMIC CDC-developed language using 4th order Runge-Kutta method with variable step-size. Rigid coding requirement but inexpensive to run.

PROSE A simulation language available on the Control Data Cybernet. It can handle discrete, continuous, or mixed simulation problems.

(3) Simulation Language mainly for the solution of partial differential equations.

PDEL Linear or nonlinear elliptic and parabolic partial differential equations in one to three-space dimensions, and hyperbolic equations in one-dimensional space.

LEANS Solution for elliptic, parabolic, or hyperbolic PDE in one- to three-dimensional spaces and in orthogonal, cylindrical, or spherical coordinate systems.

DSS Similar to LEANS except for hyperbolic equations. Some of the solutions use different methods.

PDLAN System of parabolic PDE and mainly used in meteorology.

There is also a group of HOL simulation language for discrete system simulation. Among them are SIMSCRIPT, GASP, CSL, SIMULA, and ASPOL. Some of them are supported at Pitt. However, they are outside the scope of this book.

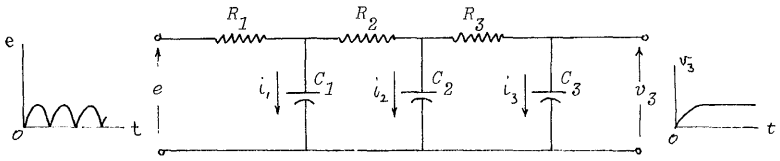
CSMP EXAMPLES

5.15 CSMP Examples

Several examples will be given here to illustrate the applications and procedures of CSMP simulation.

Example 1

A typical DC power supply is based on a full-wave rectified sine wave connected to a RC-network as a filter. One such circuit is shown below:



Let v_1, v_2 and v_3 be the voltages across the capacitors C_1, C_2, and C_3 respectively. Let i_1, i_2, and i_3 be the respective capacitor current.

For a particular case study, we assume the following parameters:

R1=10.5 ohms, R2=3.5 ohms, R3=100.4 ohms

C1=45 mfds, C2=33 mfds, C3=202 mfds

Initital conditions: V10=0, V20=0, V30=0

Input: full-wave rectified sine wave, 1000 Hz with 5-volt peak

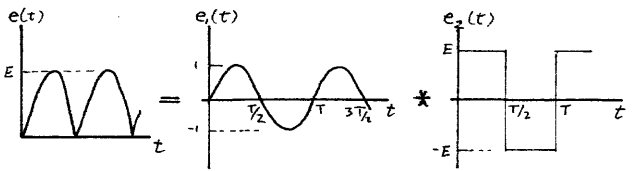
The problem analysis and modeling formulation is shown on the next page.

Capacitor currents:	Change to integral form:	Their CSMP statements:
$i_1 = C_1 \frac{dv_1}{dt}$	$v_1 = \int \frac{i_1}{C_1} dt$	V1 = INTGRL(V10,I1/C1)
$i_2 = C_2 \frac{dv_2}{dt}$	$v_2 = \int \frac{i_2}{C_2} dt$	V2 = INTGRL(V20,I2/C2)
$i_3 = C_3 \frac{dv_3}{dt}$	$v_3 = \int \frac{i_3}{C_3} dt$	V3 = INTGRL(V30,I3/C3)

By Kirchoff's Law of Voltage:

$v_2 = R_3 i_3 + v_3$	$i_3 = (v_2 - v_3)/R_3$	I3 = (V2-V3)/R3
$v_1 = R_2 (i_2 + i_3) + v_2$	$i_2 = (v_1 - v_2 - R_2 i_3)/R_2$	I2 = (V1-V2-R2*I3)/R2
$e = R_1 (i_1 + i_2 + i_3) + v_1$	$i_1 = (e - v_1 - R_1 i_2 - R_1 i_3)/R_1$	I1 = (E-V1-R1*I2-R1*I3)/R1

These are six equations for the circuit model, and six dependent variables are three capacitor currents and three capacitor voltages. Note that the model may be reduced to three equations only, containing as dependent variables of either three capacitor voltages or three capacitor currents. The reduction is uncessary in the CSMP simulation, so we will just leave them in the unreduced form.



The full-wave rectified sine wave may be synthesized by a multiplication of two time fuinctions:

$$e(t) = e_1(t) * e_2(t)$$

where e_1 = sine wave, and e_2 = square with Frequency and phase relations as shown.

The CSMP program for Example 1 is listed below:

```
INITIAL
     TITLE EXAMPLE 1:  FULL-WAVE RECTIFIER FILTER STUDY
     PARAMETER R1=10.5, R2=3.5, R3=100.4, ...
               C1=.000045,C2=.000033,C3=.000202
     INCON V10=0.0, V20=0.0, V30=0.0

DYNAMIC
* FULL-WAVE RECTIFIED SINE WAVE, 1000 HZ, 5 VOLT PEAK
     E=(10.0*PULSE(.0005,IMPULS(0.0,.001))-5.0)* ...
        SINE(0.0,6.283185E+3,.001)
* CIRCUIT MODEL EQUATIONS
     V3=INTGRL(V30,I3/C3)
     V2=INTGRL(V20,I2/C2)
     V1=INTGRL(V10,I1/C1)
     I3=(V2-V3)/R3
     I2=(V1-V2-R2*I3)/R2
     I1=(E-V1-R1*I2-R1*I3)/R1

TERMINAL
* T=0 TO T=.002 IN FINE INCREMENTS
     TIMER DELT=.00001,OUTDEL=.00005,FINTIM=.002
     METHOD RKSFX
     PRTPLOT E,V3
* T=.002 TO T=0.4 IN COARSE INCREMENTS
     CONTINUE
     TIMER OUTDEL=.01,FINTIM=.4
     END
     STOP

ENDJOB
```

The output is produced in two periods: in fine increments of 0.00005 second for t=0 to t=0.002 second, and then in coarse increments of 0.01 from t=0.002 to t=0.4 second. The purpose of the coarse increments in this study is to find the level of a steady state DC output, which is obtained from the printplots as 3.184 volts. The output printout of the CSMP run contains many parts: (1) a listing of the CSMP program, (2) a liting of the TIMER variables, (3) problem durations, (4) range of dependent variables (maxmimum and minimum values with respective time values), and (5) printplots as specified. The output was rather voluminous. Only the filter output V3 printplots are reproduced here for illustration, as shown in Figure 5.2. The output data are also captured on a disk file (see Section 5.12), which are then plotted on a Calcomp plotter. These plots are shown in Figures 5.4(a) on page 220.

FILTER OUTPUT PRINTPLOT FROM T=0 TO T=0.002 SECONDS:

```
                        Minimum              V3     versus TIME          Maximum
                        0.0000E+00                                     1.6391E-01
 TIME          V3               :                                              :
0.0000E+00  0.0000E+00  +
5.0000E-05  6.3121E-06  +
1.0000E-04  8.5250E-05  +
1.5000E-04  3.6753E-04  +
2.0000E-04  9.9247E-04  +
2.5000E-04  2.0717E-03  +
3.0000E-04  3.6685E-03  -+
3.5000E-04  5.7878E-03  -+
4.0000E-04  8.3743E-03  --+
4.5000E-04  1.1318E-02  ---+
5.0000E-04  1.4463E-02  ----+
5.5000E-04  1.7641E-02  -----+
6.0000E-04  2.0790E-02  ------+
6.5000E-04  2.3988E-02  -------+
7.0000E-04  2.7357E-02  --------+
7.5000E-04  3.1008E-02  ---------+
8.0000E-04  3.5010E-02  ----------+
8.5000E-04  3.9377E-02  -----------+
9.0000E-04  4.4060E-02  ------------+
9.5000E-04  4.8960E-02  -------------+
1.0000E-03  5.3930E-02  --------------+
1.0500E-03  5.8808E-02  ---------------+
1.1000E-03  6.3542E-02  ----------------+
1.1500E-03  6.8216E-02  -----------------+
1.2000E-03  7.2959E-02  ------------------+
1.2500E-03  7.7889E-02  -------------------+
1.3000E-03  8.3081E-02  --------------------+
1.3500E-03  8.8553E-02  ---------------------+
1.4000E-03  9.4263E-02  ----------------------+
1.4500E-03  1.0012E-01  ------------------------+
1.5000E-03  1.0597E-01  -------------------------+
1.5500E-03  1.1167E-01  ---------------------------+
1.6000E-03  1.1716E-01  ----------------------------+
1.6500E-03  1.2254E-01  -----------------------------+
1.7000E-03  1.2793E-01  ------------------------------+
1.7500E-03  1.3346E-01  -------------------------------+
1.8000E-03  1.3921E-01  ---------------------------------+
1.8500E-03  1.4519E-01  ----------------------------------+
1.9000E-03  1.5137E-01  -----------------------------------+
1.9500E-03  1.5766E-01  ------------------------------------+
2.0000E-03  1.6391E-01  -------------------------------------+
```

FILTER OUTPUT PRINTPLOT FROM T=0.002 TO 0.4 SECOND:

```
                        Minimum              V3     versus TIME          Maximum
                        1.6391E-01                                     3.1837E+00
 TIME          V3               :                                              :
2.0000E-03  1.6391E-01  +
1.2000E-02  1.2151E+00  ----------------+
2.2000E-02  1.9041E+00  ------------------------+
3.2000E-02  2.3520E+00  ------------------------------+
4.2000E-02  2.6430E+00  --------------------------------+
5.2000E-02  2.8322E+00  ----------------------------------+
6.2000E-02  2.9552E+00  -----------------------------------+
7.2000E-02  3.0352E+00  ------------------------------------+
8.2000E-02  3.0871E+00  -------------------------------------+
9.2000E-02  3.1209E+00  -------------------------------------+
1.0200E-01  3.1428E+00  --------------------------------------+
1.1200E-01  3.1571E+00  --------------------------------------+
1.2200E-01  3.1664E+00  --------------------------------------+
1.3200E-01  3.1724E+00  --------------------------------------+
1.4200E-01  3.1763E+00  --------------------------------------+
1.5200E-01  3.1789E+00  --------------------------------------+
1.6200E-01  3.1805E+00  --------------------------------------+
1.7200E-01  3.1816E+00  --------------------------------------+
1.8200E-01  3.1823E+00  --------------------------------------+
1.9200E-01  3.1828E+00  --------------------------------------+
2.0200E-01  3.1830E+00  --------------------------------------+
2.1200E-01  3.1832E+00  --------------------------------------+
2.2200E-01  3.1834E+00  --------------------------------------+
2.3200E-01  3.1834E+00  --------------------------------------+
2.4200E-01  3.1835E+00  --------------------------------------+
2.5200E-01  3.1835E+00  --------------------------------------+
2.6200E-01  3.1836E+00  --------------------------------------+
2.7200E-01  3.1836E+00  --------------------------------------+
2.8200E-01  3.1836E+00  --------------------------------------+
2.9200E-01  3.1836E+00  --------------------------------------+
3.0200E-01  3.1836E+00  --------------------------------------+
3.1200E-01  3.1836E+00  --------------------------------------+
3.2200E-01  3.1836E+00  --------------------------------------+
3.3200E-01  3.1836E+00  --------------------------------------+
3.4200E-01  3.1836E+00  --------------------------------------+
3.5200E-01  3.1836E+00  --------------------------------------+
3.6200E-01  3.1836E+00  --------------------------------------+
3.7200E-01  3.1836E+00  --------------------------------------+
3.8200E-01  3.1836E+00  --------------------------------------+
3.9200E-01  3.1836E+00  --------------------------------------+
```

Figure 5.2 Printplots Output for V3 versus TIME

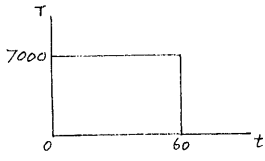
Example 2

A small rocket has an initial weight of 3000 pounds, including 2400 pounds of fuel. It is fired vertically upward. The rocket burns fuel at a constant rate of 40 pounds/second, which produces a constant thrust of 7000 poinds.

The drag force acts in the opposite direction of the motion, and it is obtained by two simplifying assumptions: (1) It is proportional to the square of velocity (D=Kv). (2) The coefficient of aerodynamic resistance K has an average value of 0.008 lb-sec /ft . Thus,

$$D = 0.008 \left(\frac{dy}{dt}\right)^2 \quad \text{for } y>0$$

Therefore, the thrust T may be specified in the following way:



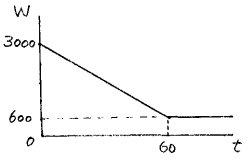
T = 7000 for $0 \leqslant t \leqslant 60$

T = 0 for $t > 60$

In CSMP statement, it may be written as:

```
T = 7000.0*(STEP(0.0)-STEP(60.0))
```

The weight of the rocket, W, is also a time-varying function:

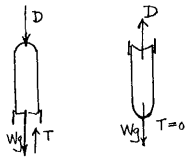


W = 3000 -40 t for $0 \leqslant t \leqslant 60$

X = 600 for $t > 60$

In terms of CSMP, W may be written as:

```
W = 3000.0-40.0*RAMP(0.0)+40.0*RAMP(60.0)
```



Going up Coming down

Consider the rocket as a free body. The two diagrams indicate the forces acting on the free body. One diagram is for the rocket on its way UP, and the other is for it on its way DOWN. The difference is in the direction of the drag force, which is always opposite to the motion. Let the "y" be positive in the upward direction.

The net force UPWARD = Tg - Wg $\pm$ D

where "+" sign is for downward leg, and "-" sign is for the upward leg of the journey.

By Newton's Law, $$W \frac{d^2y}{dt^2} = Tg - Wg \pm 0.008\, g \left(\frac{dy}{dt}\right)^2$$

or, $$\frac{d^2y}{dt^2} = \frac{Tg}{W} - g \pm \frac{.008\, g \left(\frac{dy}{dt}\right)^2}{W}$$

Now reduce the second-order equation to a system of two simultaneous first-order equations:

Let $vel = \frac{dy}{dt}$

Then, $\frac{d(vel)}{dt} = F$

where $F=Tg/W-g\pm0.008g(vel)^2/W$

and the initial conditions are:

$y(0)=0$ and $vel(0)=0$

The CSMP statements are:

```
T = INTGRL(0.0,VEL)
VEL = INTGRL(0.0,EXPR)
EXPR = THRUST*G/WEIGHT - G -- ...
  SIGN(1.,VEL)*(.008*G*VEL*VEL)/WEIGHT
```

where SIGN is a FORTRAN function.

The complete CSMP program for Example 2 is listed below:

```
INITIAL
   TITLE EXAMPLE 2:  ROCKET PROBLEM
   CONSTANT G=32.2, K=0.008

DYNAMIC
   THRUST=7000.0*(STEP(0.0)-STEP(60.0))
   WEIGHT=3000.0-40.0*RAMP(0.0)+40.0*RAMP(60.0)
   EXPR=THRUST*G/WEIGHT - G - SIGN(1.0,VEL)*(K*G*VEL*VEL)/WEIGHT
   Y=INTGRL(0.0,VEL)
   VEL=INTGRL(0.0,EXPR)

TERMINAL
   TIMER DELT=.03125,OUTDEL=10.0,FINTIM=60.0
   METHOD RKSFX
   PRTPLT Y,VEL

   CONTINUE
   TIMER OUTDEL=1.0,FINTIM=70.0

   CONTINUE
   TIMER OUTDEL=0.1,FINTIM=120.0

   FINISH VEL=-10.0

   END
   STOP

ENDJOB
```

The composite diagrams on the next page are the printer-output of the simulation. From the VEL-TIME and the Y-TIME plots, the rocket reaches a maximum velocity of 890.39 ft/sec at about 60.0 seconds, and reaches a maximum height of 43,347 ft at 70.8 seconds.

Printplots of Velocity versus TIME:

```
 TIME          VEL          :                                        :
0.0000E+00    0.0000E+00    +
1.0000E+01    4.2446E+02    ----------+
2.0000E+01    6.8238E+02    ----------+
3.0000E+01    7.8040E+02    ----------+
4.0000E+01    8.2477E+02    ----------+
5.0000E+01    8.5872E+02    ----------+
6.0000E+01    8.9039E+02    ----------(+)   Max velocity

6.0000E+01    8.9039E+02    ----------+
6.1000E+01    6.1995E+02    ----------+
6.2000E+01    4.6363E+02    ----------+
6.3000E+01    3.5941E+02    ----------+
6.4000E+01    2.8316E+02    ----------+
6.5000E+01    2.2351E+02    ----------+
6.6000E+01    1.7436E+02    ----------+
6.7000E+01    1.3207E+02    ----------+
6.8000E+01    9.4350E+01    ----------+
6.9000E+01    5.9576E+01    ----------+
7.0000E+01    2.6545E+01    +

7.0000E+01    2.6545E+01    ----------+
7.0100E+01    2.3298E+01    ----------+
7.0200E+01    2.0058E+01    ----------+
7.0300E+01    1.6823E+01    ----------+
7.0400E+01    1.3593E+01    ----------+
7.0500E+01    1.0367E+01    ----------+
7.0600E+01    7.1439E+00    ----------+
7.0700E+01    3.9226E+00    ----------+
7.0800E+01    7.0229E-01    ----------+
7.0900E+01   -2.5176E+00    ----------+
7.1000E+01   -5.7369E+00    ----------+
7.1100E+01   -8.9545E+00    ----------+
7.1150E+01   -1.0562E+01    +
```

Printplots of Height versus TIME:

```
 TIME          Y            :                                        :
0.0000E+00    0.0000E+00    +
1.0000E+01    2.1767E+03    ----------+
2.0000E+01    7.8895E+03    ----------+
3.0000E+01    1.5284E+04    ----------+
4.0000E+01    2.3328E+04    ----------+
5.0000E+01    3.1749E+04    ----------+
6.0000E+01    4.0495E+04    ----------+

6.0000E+01    4.0495E+04    +
6.1000E+01    4.1236E+04    ----------+
6.2000E+01    4.1772E+04    ----------+
6.3000E+01    4.2181E+04    ----------+
6.4000E+01    4.2500E+04    ----------+
6.5000E+01    4.2752E+04    ----------+
6.6000E+01    4.2951E+04    ----------+
6.7000E+01    4.3103E+04    ----------+
6.8000E+01    4.3216E+04    ----------+
6.9000E+01    4.3293E+04    ----------+
7.0000E+01    4.3336E+04    ----------+

7.0000E+01    4.3336E+04    +
7.0100E+01    4.3338E+04    ----------+
7.0200E+01    4.3341E+04    ----------+
7.0300E+01    4.3342E+04    ----------+
7.0400E+01    4.3344E+04    ----------+
7.0500E+01    4.3345E+04    ----------+
7.0600E+01    4.3346E+04    ----------+
7.0700E+01    4.3347E+04    ----------+
7.0800E+01    4.3347E+04    ----------(+)   Max
7.0900E+01    4.3347E+04    ----------+     height
7.1000E+01    4.3346E+04    ----------+
7.1100E+01    4.3346E+04    ----------+
7.1150E+01    4.3345E+04    ----------+
```

Figure 5.3 Composite Printplots of Example 2

Example 3

We now conclude with an example which would not be normally considered as an engineering study.

Let us consider a dynamic model of influenza epidemic. We first make three assumptions: (1) The disease spreads when a susceptible person comes in contact with a infected person. (2) A person who recovers from the influenza is normally immune for a certain period of time. (3) Immunity is ultimately lost, and the person becomes susceptible again to the disease.

In addition, for the sake of model simplicity, we assume no birth and no death occur to the group of population under study.

So the population under consideration is composed of three groups:

Group	CSMP Variable	
Susceptible population	SUSP	in no. of persons
Infected population	INFP	in no. of persons
Immuned population	IMMP	in no. of persons

Define the rates in the following manner:

IR = Infection rate, no. of persons becoming infected each day
RR = recovery rate, no. of persons recovered each day
LR = loss-of-immunity rate, no. of persons/day who loses immunity

Thus, $\Delta(SUSP) = (LR - IR)\ \Delta t$
= increase of susceptible persons per day

$$\frac{d(SUSP)}{dt} = LR - IR$$

similarly, $$\frac{d(INFP)}{dt} = IR - RR$$

$$\frac{d(IMMP)}{dt} = RR - LR$$

There are three other equations, each defining the relations of the rates IR, RR, LR. IR is proportional to SUSP and to INFP, and therefore proportional to their product. Thus,

$$IR = K * (SUSP) * (INFP)$$

If the disease requires (PD) days to run its course, then

$$RR = (INFP)/PD$$

If the period of immunity lasts for PIMM days, then

$$LR = (IMPP)/PIMM$$

These six equations constitute the dynamic model of the influenza epidemic.

Now let us study one particular case of dynamic simulation. Suppose we have a population of 1000 persons. On day 0, one person is sick, 999 persons healthy, and nobody immunized. This gives us three initial conditions:

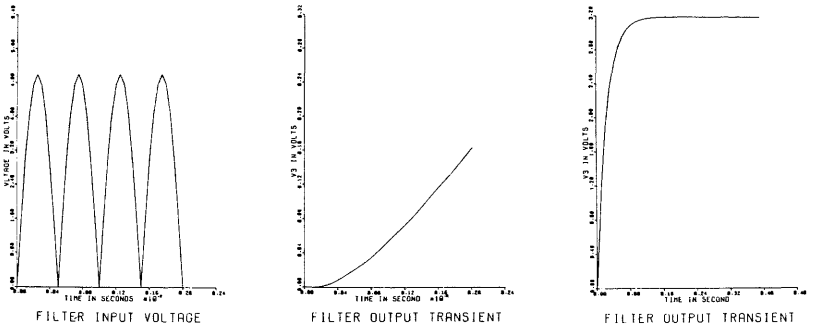
SUSP(0) = 999, INFP(0) = 1, IMMP(0) = 0

For the last three equations, we assume the following parameters:

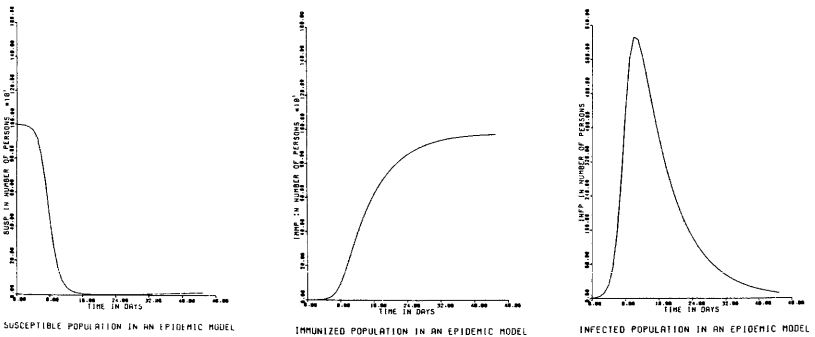
K = 0.001, PD = 8 days, and PIMM = 1000 days.
The CSMP program for this example is listed below:

```
INITIAL
      INCON  SUSO=999.0,INFO=1.0,IMMO=0.0
      PARAMETER K=0.001,PD=8.0,PIMM=1000.0

DYNAMIC
      IR=K*SUSP*INFP
      RR=INFP/PD
      LR=IMMP/PIMM
      SUSP=INTGRL(SUSO,LR-IR)
      INFP=INTGRL(INFO-IR-RR)
      IMMP=INTGRL(IMMO,RR-LR)

TERMINAL
      TIMER OUTDEL=1.0,FINTIM=45.0
      PRTPLT SUSP,INFP,IMMP
      END
      STOP

ENDJOB
```

The plots are made on the CalComp plotter with the "captured" data from the CSMP run, and they are shown in Figure 5.4(b) on the next page. From these plots and those printouts from the CSMP run, the epidemic simulation concludes that the crisis of the epidemic occurs on the 10th day when 613 out of 1000 persons are sick. After that, crisis passes and the infected population is reduced to 13 persons on the 45th day.



(a) CalComp Plotter Output of Example 1

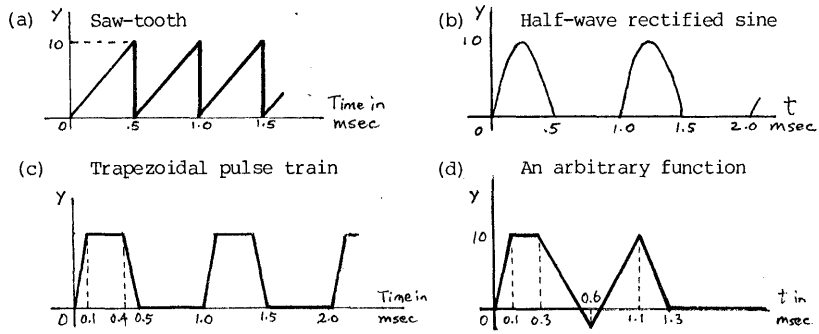


(b) CalComp Plotter Output of Example 3

Figure 5.4 CalComp Plotter Output of CSMP Examples

EXERCISES

1. Write CSMP functions for the the following time-functions, and verify by CSMP printouts:



2. The following differential equations have known analytical solutions. Use CSMP to obtain their computer solutions and verify them with the given analytical solutions. Specify different methods and compare the accuracy of the results.

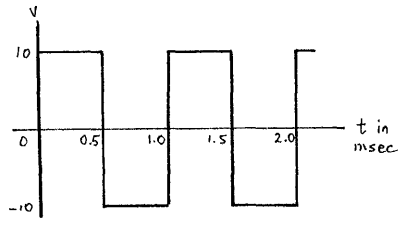
(a) $y' + 3y = x + e^{-2t}$, $y(0) = 0$
Solution: $y = -\frac{8}{9}e^{-3t} + \frac{t}{3} - \frac{1}{9} + e^{-2t}$

(b) $y' = 1 + t + y^2 + ty^2$, $y(0) = 0$
Solution: $y = \tan(t+t^2/2)$

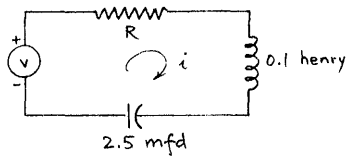
(c) $(1-t)y'' + ty' - y = 2(t-1)^2 e^{-t}$, $y(0)=-0.5$, $y'(0)=0$
for $0<t<1$
Solution: $y = -e^t + 2t - \frac{1}{2}e^{-t}(2t-1)$

(d) $y''' - y' = t$, $y(0)=6.0$, $y'(0)=0$, $y''(0)=1.0$
Solution: $y = 1 + 2e^t + 3e^{-t} - \frac{x^2}{2}$

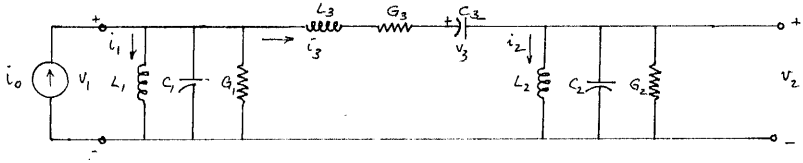
3. A square wave of 10 volts amplitude is applied to a RLC circuit as shown. By CSMP, find the current as a function of time.



(a) R = 1000 ohms
(b) R = 400 ohms
(c) R = 100 ohms



4. The circuit shown is a sixth-order circuit, and its behavior is given by a matrix equation:*

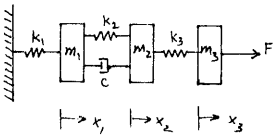


$$\begin{bmatrix} v_1' \\ v_2' \\ v_3' \\ i_1' \\ i_2' \\ i_3' \end{bmatrix} = \begin{bmatrix} -\frac{G_1}{C_1} & 0 & 0 & -\frac{1}{C_1} & 0 & -\frac{1}{C_1} \\ 0 & -\frac{G_2}{C_2} & 0 & 0 & -\frac{1}{C_2} & \frac{1}{C_2} \\ 0 & 0 & 0 & 0 & 0 & \frac{1}{C_3} \\ \frac{1}{L_1} & 0 & 0 & 0 & 0 & 0 \\ 0 & \frac{1}{L_2} & 0 & 0 & 0 & 0 \\ \frac{1}{L_3} & -\frac{1}{L_3} & -\frac{1}{L_3} & 0 & 0 & -\frac{R_3}{L_3} \end{bmatrix} \cdot \begin{bmatrix} v_1 \\ v_2 \\ v_3 \\ i_1 \\ i_2 \\ i_3 \end{bmatrix} + \begin{bmatrix} \frac{i_o}{C_1} \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

Find the current i_1, i_2, and i_3 as functions of time by CSMP simulation.
Use: G_1 = .001 mho, G_2 = .002 mho, and G_3 = .01 mho;
L_1 = .1 h, L_2 = .5 h, and L_3 = 1.0 henry;
C_1 = 10 mfds, C_2 = 25 mfds, and C_3 = 100 mfds.
i_o = 10 milliamperes.

5. A ball is rolling off the edge of a flat roof 30 feet from the ground. When it leaves the edge, it was travelling at a speed of 5 feet per second. When the ball hits the ground, certain fixed percentage of kinetic energy was absorbed. That percentage depends on the type of ball: superball 5%, rubber ball 15%, basket ball 55%, iron ball 90%. Assume the angle of incidence is equal to the angle of reflection when the ball bounces up. Find the x- and the y-component of the velocity of the bouncing ball after it leaves the roof. Do this for each of four kinds of balls.

6. A sinusoidal force (F=12 sin(4t)) applied to the following mechanical system. Find the displacements x_1, x_2 and x_3 at TIME=2.0. (Answer: -0.2075, -0.2707, 0.4517)



$$m_1x_1'' + (k_1+k_2)x_1 + cx_1' - k_2x_2 - cx_2'' = 0$$

$$m_2x_2 + (k_2+k_3)x_2 + cx_2' - k_2x_1 - cx_1' - k_3x_3 = 0$$

$$m_3x_3'' + k_3x_3 - k_3x_2 = 12\sin(4t)$$

where: m_1= 2.3, m_2=3.4, m_3=0.9

k_1=19.0, k_2=45.0, k_3=12.0, c=0.8

*BASIC CIRCUIT THEORY, by L. P. Huelsman, Prentice-Hall, Inc., 1972; pp.362-366.

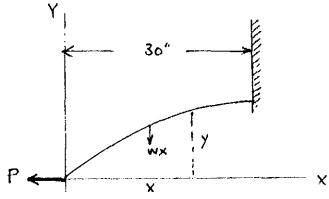
7. A control system has a transfer function $1/(s^2+s+1)$.

   (a) Find the transient response to a unit step function.

   (b) Find the frequency response over the range of 0.1 to 10 radians/second.

8. Suppose a cantilever beam of length 30 inches and weighing 10 lb/in is subjected to a horizontal tensile force of 100 lb applied at the free end. Taking the origin at the free end and the y-axis positive upwards, the equation of the beam is:

$$EIy'' = P\,y - w\,x^2/2$$



   where E = Young's modulus, 30E+06 psi
   I = Moment of inertia = 0.01042 in
   w = linear weight per length
   = 10 lb/in
   P = 100 lb
   y'= deflection of beam

   The beam is so placed that at x=0, y(0)=0, and y'(30)=0. Find the maximum deflection at the end of the beam.

9. When a bomb is dropped from an airplane, it encounters an air resistance proportional to the square of the velocity, and acquires a velcity of 125 ft/sec in falling a distance of of 343 feet, find the time elapsed and the limiting velocity.

Note to the Instructor:

The purpose of this group of exercise problems is to familiarize with the CSMP programming and execution. The derivation of models, however important, is not the main goal of these exercises. In this context and depending on the background of the class, additional problems may be found from many standard texts in physics, circuits, mechanics, control systems, etc., where dynamic behaviors are discussed.

REFERENCES

1. IBM GH20-0367-4: SYSTEM/360 CONTINUOUS SYSTEM MODELING PROGRAM USER'S MANUAL, Program No. 360A-CX-16X; Fifth Edition, 1972.

2. A GUIDE TO USING CSMP - THE CONTINUOUS SYSTEM MODELING PROGRAM, Frank Speckhar and Walter Green, Prentice-Hall, Inc.; 1976.

3. APPLIED NUMERICAL METHOD FOR DIGITAL COMPUTATION WITH FORTRAN AND CSMP, Second Edition, M. L. James, G. M. Smith and J. C. Wolford, chapter 6, pp. 569-636, Harper and Row Company; 1977.

4. DEC-10 System Help File: SYS:CSMP.HLP, Computer Center, University of Pittsburgh, Pittsburgh, Pennsylvania; 1978.

5. MODELING AND SIMULATION BY CSMP, Class Notes for EE45 (Computer Application I), T. W. Sze, University of Pittsburgh, Pittsburgh, Pennsylvania; 1980.

6. A CSMP PRIMER, Class Notes for EE45 (Computer Applications I), T. W. Sze, University of Pittsburgh, Pittsburgh, Pennsylvania; 1980.

7. ENGINEERING SIMULATION USING SMALL SCIENTIFIC COMPUTERS, Manesh J. Shah, Prentice-Hall, Inc., Englewood Cliffs, NJ; 1976.

8. MATHEMATICAL MODELING WITH COMPUTERS, Samuel L. S. Jacoby and Janusz S. Kowalik, Prentice-Hall, Inc., Englewoord Cliffs, New Jersey; 1980.

CHAPTER 6

A PRIMER OF COMPUTER GRAPHICS WITH DEC-10

Computer graphics is defined as "the art and the science of producing graphical images with the aid of a computer." (Reference 1)

For many years, the computer field has tried to break out the bottleneck of computer-human communications by using pictures rather than printouts. The idea is an old one. After all, "a picture is worth ten thousand words." Hence, earliest use of computer graphics was simply to present data in graphical form. But earlier computer graphics was very expensive. Hardware was costly, but far worse was the software cost. They were machine-dependent and programming was on the assembler language level. Much of the acceptance of the present-day computer graphics is attributed to these developments:

(1) Development of user-accessible software of computer graphics in the forms of FORTRAN callable subroutines. This puts the application of computer graphics at the user's hand, rather than at the hand of a professional programmer.

(2) The time-sharing mode of computer operations is further enhanced by the computer graphics with its clarity of man-machine interaction.

(3) LSI and microprocessor development has rapidly brought down the cost of computer graphics hardware.

Computer graphics, of course, can do much more than plotting nowadays. In fact, it becomes a branch of computer processing that has most caught the imagination of the non-computer world. We find its applications in computer-aided design and manufacturing, simulation of training environment, and science fiction motion pictures and many other widely different fields.

6.1 Computer Graphics and Computer Graphics Devices

Using the definitions of Ivan E. Sutherland, a pioneer in computer graphics, there are two types of computer graphics systems now in common use: rasterized systems and calligraphic systems.

The rasterized systems make picture the same way a television set does. It is drawn in a fixed sequence, usually from the left to the right and from top to bottom. It has the advantage of simpler and less expensive implementation,

as it is compatible with the conventional mass-produced display devices such as terminals, printers or video display sets. Implementation of computer graphics of this type, however, requires much computer effort in sorting the display information in the sequential display, a process called rasterization. If the display data are altered, the graphic data would require another rasterization.

The calligraphical systems, on the other hand, will construct a picture in any sequence of plotting given by the computer. The picture-drawing element is a mechanically moved pen of a plotter or an electronically controlled electron beam in a cathode-ray-tube (CRT) display. The pen or the electron beam is moved from one point to another at the command of the computer. When the pen is held down on paper, or when the beam is turned on, it traces a line on the paper or the screen during its movement. When the pen is up, or when the beam is blanked, it moves the picture-drawing element to another point without leaving a trace. This is exactly the same process of preparing a line drawing manually. The advantage of using a calligraphic device is that the information on sequence of tracing can be stored in the computer in any order. Any alteration of the display data can be simply made by revising the stored data.

The decade of 1970's has been a period of time that brought in the large-scale integrated circuit (LSI) into digital electronics. As a result, the computer graphics enjoyed a phenominal growth both in hardware development and in applications. Unfortunately, there has been very little coordination in the growth. As a result, different types of graphic hardwares use different software packages, most of which are not compatible to each other. Although energetic efforts have been made in the United States and abroad to standardize the graphics field (References 3 and 4), progress has been slow.

In this chapter, the introduction to the computer graphics will be by necessity restricted to the available hardware/software packages for the DEC-10 at a local installation. For this reason, readers should check with their own installation regarding the available graphic hardware and software. The hardware graphic devices are listed as follows:

1. Rasterized devices

   Terminals
   Line printers
   Image display devices

2. Calligraphic devices

   Plotters
   Graphics terminals

For the graphics usage on DEC-10, the materials will be presented in three parts. In each of these parts, certain devices in the above list will be employed. The parts are:

1. Graphing and plotting

2. General graphics

3. 3-Dimensional graphics

GRAPHING AND PLOTTING

Let us consider the steps we take in plotting a graph after the x-y array data have been determined:

(1) Assign dependent and independent variables respectively to the x and y axis.

(2) Determine the range of the ordinates and abscissas.

(3) Since the plotting field is usually fixed, we map the range of abscissa and ordinates onto the fixed paper ranges. This is called virtual graphics, a term we will use very often later. Its explanations will also be left for latter treatment.

(4) After the range is determined, choose a scale factor.

(5) As the independent variable is incremented, we determine a dependent variable value for that point, and then determine by scale factor the position of that point on the plot. When this step is repeated as x-variable incremented, points are joined by either straight lines, or a smooth (French) curve, or a best-fit curve.

We will now consider the plotting and graphings on both the rasterized and the calligraphic devices.

6.2 Plotting on a Terminal or Printer

Both the terminals and the printers are rasterized devices. There is only one direction the paper may be advanced. We are excluding certain more costly terminals that has paper movement control (forward and backward) built into the device.

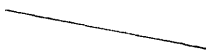
Plotting and graphing on a terminal or a printer is basically the sequential outputing of a character-print line. Let us consider a typical case.

Consider a FORTRAN format of (101A1) for use by a character array LI(I) with I runs from 1 to 101. Thus there are 101 print positions from 0 to 100. Now, the data point values are generated and the maximum and the minimum values of the ordinates can be determined. The range for the ordinates (Ymax-Ymin) can then be calculated. With the bounds of the ordinates established, any value within the bound can be mapped into a proportional value within the print position bounds of 0 to 100. This will then establish a print position for that particular ordinate. Thus, if the LI-array is initialized as blank (Octal word code "200000000000), the LI-element at the print position, say at position KK, is set to an ASCII character of a plot symbol, such as a "*". Thus, LI(KK)='*', while LI-element is blank everywhere else. When this array is now printed with a (101A1) format, a symbol of "*" is printed as a scaled ordinate. The LI-array is then re-initialized to all blanks, and the process is repeated for the next ordinate value.

There are many amenities that can be built into such a plotting routine. For example:

a. Scale factor printed along with the graph;
b. Choice of scale factor limited to 1,2, or 5 or their integer (positive or negative) powers of ten;
c. Graph with or without grid lines;
d. Multiple curves on one plot;
e. Cartesian plots, semilog plots, log plots, or polar plots;
f. Curves occupy 60-90% of graphing space for neat looking plot.

Terminal or printer plots have the advantages of being inexpensive and fast turn-around. Unlike a plotter plot, they can be immediately produced on the user's terminal or an accessible printer, and the turn-around time is very short. However, there is at most about 101 print positions, so the resolution of a plot is not good. The worst case would be a curve that is almost horizontal or almost vertical. An almost horizontal curve will have an appearance of a staircase such as shown below:



A Straight Line

```
**********
          **********
                    **********
                              **********
```

Line as appeared on a Printer Plot

Thus, the chief usefulness of a terminal or printer plot is to provide a quick and inexpensive way of graphical output, when high resolution and polished drafting quality are not as important requirements.

Consequently, it would not be cost effective for any user to design and implement his own plotting routine, except as a format-exercise, because there are so many already available, and because its plot quality is poor.

For the remaining portion of the discussion of plotting and graphing, materials will be devoted to the presentation of several typical available plotting softwares that are either FORTRAN-callable subroutines, or stand-alone interactive programs. In both cases, the users are spared from the drudgery of laborious construction of axes, determination of scale factors, marks on the axes, design of mapping formula, and so on.

For the terminal or printer plot routines, several FORTRAN subroutines in the Engineering Program Library will be presented below. These graphical routines are available for user's call under the collective name of ENG:GRAPH.REL. Thus the standard way of calling a library routine in conjuction of your main program (assuming named as PRGM.FOR) is to issue a monitor command of:

```
.EXECUTE PRGM, ENG:GRAPH/LIB
```

(1) The PLOT8 Subroutine

The subroutine call is: CALL PLOT8(Y,NF,NP,X1,XINC)

where the parameters are defined as follows:

Y a 2-dimensional real array Y(I,J) for the ith function and jth value. Maximum size is 8 functions. The calling program must have the Y-array dimensioned at Y(NF,NP). The Y-array will be altered upon the return of this subroutine.

NF number of functions to be plotted; max NF= 8.

NP number of points to be plotted for each function. Max=151.

X1 value of the first abscissa

XINC value of X-increment. This subroutine is for a plot that the x-increment is constant.

A terminal or printer-constructed plot is produced. The positive direction of the independent variable is taken as downward on the printed page. The symbols for up to 8 curves are assigned as:

Function:	1	2	3	4	5	6	7	8
Symbol:	*	+	#	x	o	.	-	=

If points of different functions occupy the same print position, the joint symbol at that print position is a "$". For convenience, the numercial values of the first function are printed along the left edge.*

Example: Graph three functions, whose data are stored respectively in DA.DAT, DB.DAT, and DC.DAT. Each data file contains a Y-array of the same number of elements (NP=46). Also, X1=0, and XINC=1.0. The program (named as SAMPLE.FOR) that calls the subroutine is listed below:

```
      REAL Y(3,46)
      CALL IFILE(1,'DA')
      CALL IFILE(2,'DB')
      CALL IFILE(3,'DC')
50    FORMAT(F)
      DO 10 I=1,3
         READ(I,50)(Y(I,J),J=1,46)
         CALL PLOT8(Y,3,46,0.0,1.0)
10    CONTINUE
      END
```

The execution command is: ".EXECUTE SAMPLE,ENG:GRAPH/LIB" The output of this program is shown on Figure 6.1 on the next page.

---

*If this output is produced on a terminal, a right margin setting should be preset by a monitor command of "TTY WIDTH 132".

```
                The Scale Factor of Ordinate: 1 Division= 0.20000E+02
                The Scale Factor of Abscissa: 1 Division= 0.10000E+01
                        First Abscissa Value:       X(1)= 0.00000E+00

          NOTE: In interpreting the plot, X-axis starts with X(1) value.
                Other X's can be computed from X(1) and abscissa scale factor.

Values of                                  (Multiply by Scale Factor  0.20000E+02)
Function 1      -10       0        10        20        30        40        50        60        70        80        90
----------      +---------+---------+---------+---------+---------+---------+---------+---------+---------+---------+
0.9990E+03      |         $                                                   *
0.9974E+03      |         $                                                   *
0.9936E+03      |         $                                                   *
0.9846E+03      |         +#                                                  *
0.9637E+03  X(  5)+       + #                                                 *
0.9175E+03      |          +  #                                              *
0.8232E+03      |          +     #                                           *
0.6642E+03      |             +        #                                     *
0.4617E+03      |                +                #*
0.2781E+03  X( 10)+                   +     *           #
0.1538E+03      |                      *  +                #
0.8353E+02      |                   *        +                #
0.4658E+02      |                 *                +                #
0.2730E+02      |                *                          +  #
0.1701E+02  X( 15)+               *                             # +
0.1134E+02      |                 *                                 #    +
0.8111E+01      |                *                                        #    +
0.6236E+01      |                *                                           #    +
0.5141E+01      |                *                                              #    +
0.4519E+01  X( 20)+               *                                                 #    +
0.4195E+01      |                *                                                    #    +
0.4070E+01      |                *                                                       #    +
0.4083E+01      |                *                                                          #    +
0.4197E+01      |                *                                                             #    +
0.4387E+01  X( 25)+               *                                                                #    +
0.4639E+01      |                *    #                                                                #    +
0.4942E+01      |                *    #                                                                     #    +
0.5288E+01      |                *    #                                                                          #    +
0.5673E+01      |                *    #                                                                               #    +
0.6092E+01  X( 30)+               *    #                                                                               #    +
0.6543E+01      |                *    #                                                                               #    +
0.7024E+01      |                *    #                                                                               #    +
0.7532E+01      |                *    #                                                                               #    +
0.8066E+01      |                *    #                                                                               #    +
0.8625E+01  X( 35)+               *    #                                                                               #    +
0.9207E+01      |                *    #                                                                               #    +
0.9811E+01      |                *    #                                                                               #    +
0.1044E+02      |                *    #                                                                               #    +
0.1108E+02      |                $                                                                                    #    +
0.1175E+02  X( 40)+               $                                                                                    #    +
0.1243E+02      |                $                                                                                    #    +
0.1313E+02      |                $                                                                                    #    +
0.1385E+02      |                $                                                                                    #    +
0.1458E+02      |                $                                                                                    #    +
0.1533E+02  X( 45)+               $                                                                                    #     +
0.1609E+02      |                $                                                                                    #     +
----------      +---------+---------+---------+---------+---------+---------+---------+---------+---------+---------+
                -10       0        10        20        30        40        50        60        70        80        90
                                           (Multiply by Scale Factor  0.20000E+02)
```

Figure 6.1 Output from the Subroutine PLOT8

```
HELP file for ENG:GRAPH Package:

**********************
*  SUBROUTINE  PLOT8  *
**********************
*
*       SUBROUTINE PLOT8(Y,NF,NP,X1,XINC)
*
*       Subroutine to plot up to "NF" curves on the same plot
*         with automatic scaling and choice of best scale factor
*
*       Y       2-dimensional real array Y(i,j) for the ith function,
*                and jth value. Max i=8, max j=151.
*       NF      number of functions to be plotted, max =8
*       NP      number of points for each function, max=151
*               Each function must have the same NP.
*       X1      first abscissa value
*       XINC    increment of X's
*
*       Ordinates of the first function will be tabulated  on  the left
*         side of the plot.  When points from different curves coincide,
*         they will be plotted as a single point marked with "$" symbol.
*
*       By T. W. Sze, December 10, 1972; single curve plot
*       Revised TWS, October 8, 1977; multiple-curve plot
*
*
*


**********************
*  SUBROUTINE XYPLOT  *
**********************
*
*       SUBROUTINE XYPLOT(X,Y,ND)
*
*       Subroutine to plot a x-y plot with automatic scaling and choice
*         of the best scale factor.  The increments of abscissa may be
*         unequal, the x-array need not be in ascending order, nor need
*         they be unique from each other.  Therefore, this routine may
*         be used to plot a multi-valued function, such as a complete
*         circle.
*
*       This is also the backbone routine for polar plots, semilog - log
*         plots. In polar plots, polar coordinates are first transformed
*         into Cartesian coordinates.  In log plots, logarithmic trans-
*         formation is done on the values first.  After transformation
*         of data, calling XYPLOT routine would produce a polar or log
*         plot.
*
*       X,Y     one-dimensional array X(i), Y(i), with max i= ND
*       ND      number of points for the function
*
*       NSX     maximum abscissa scale value used in the plot
*       NSY     maximum ordinate scale value used in the plot
*       NNP     total range of values of the acscissa  scale desired for
*               the plot.
*
*
*       By T. W. Sze, October 20, 1980
*
```

```
      SUBROUTINE PLOTS(Y,NF,NP,X1,XINC)
      REAL Y(NF,NP),YM(2)
      INTEGER L(11),LI(101),SYMBOL(8)
**  JB=blank is used in IF statement, requiring precise definition.
**  Lower case "x" code is "740000000000, left justified.
**  Lower case "o" code is "674000000000, left justified.
      DATA JN,JP,JI,JZ/'-','+','1','$'/,JB/"200000000000/
      DATA SYMBOL/'*','+',' ',"740000000000,"674000000000,
     1  '.','-','=' /
      YM(1)=-1.0E+36; YM(2)= 1.0E+36
**  To establish range of Y's
      DO 10 I=1,NF
         DO 10 J=1,NP
            YM(1)=AMAX1(YM(1),Y(I,J)); YM(2)=AMIN1(YM(2),Y(I,J))
10    CONTINUE
      RANGE=YM(1)-YM(2)
**  To establish the best scale factor for Y's
      CALL SCALE(RANGE,YM(2),SF,NS)
      WRITE(6,1000)SF; WRITE(6,1010)XINC; WRITE(6,1020)X1
      WRITE(6,1030); WRITE(6,1040)
**
**  Start plotting
**
      N=0
**  Print ordinate scale
      DO 20 I=1,11
20    L(I)=10*I-110+NS
      WRITE(6,1050)SF; WRITE(6,1060)(L(I),I=1,11)
**  Construct ordinate graph line
      ND=0
      DO 30 I=1,10
         ND=ND+1; LI(ND)=JP
         DO 30 J=1,9
            ND=ND+1; LI(ND)=JN
30    CONTINUE
      LI(101)=JP
      WRITE(6,1070)LI
      XNS=NS
      GO TO 90
**
**  Change numerical data to symbols at right print positions
**
40    DO 50 I=1,NF
         KK=Y(I,N)/SF + 101.4999 - XNS
         IF(KK.GE.101)KK=101
         IF(KK.LE.1)KK=1
         IF(LI(KK).NE.JB)LI(KK)=JZ
         IF(LI(KK).EQ.JB)LI(KK)=SYMBOL(I)
50    CONTINUE
**
**  To calculate the length of a print line
**
      DO 60  I=1,101
         IX=102-I
         IF(LI(IX).NE.JB)GOTO 70
60    CONTINUE
70    LENGTH=IX
      IF(MOD(N,5).EQ.0)GO TO 80
      WRITE(6,1080)Y(1,N),(LI(I),I=1,LENGTH)
      GOTO 90
80    WRITE(6,1090)Y(1,N),N,JP,(LI(I),I=1,LENGTH)
**
**  Reset the line to all blanks, and begin next plot line
**
90    DO 100 I=1,101
100   LI(I)=JB
      N=N+1
      IF(N.LE.NP)GOTO 40
**  End of graphing, construct bottom Y-axis
      ND=0
      DO 110 I=1,10
         ND=ND+1; LI(ND)=JP
         DO 110 J=1,9
            ND=ND+1
            LI(ND)=JN
110   CONTINUE
      LI(101)=JP
      WRITE(6,1070)LI
      DO 120 I=1,11
         L(I)=10*I-110+NS
120   CONTINUE
      WRITE(6,1110)(L(I),I=1,11); WRITE(6,1100)SF; WRITE(6,1040)
      RETURN
1000  FORMAT(/45X,42H The Scale Factor of Ordinate: 1 Division=, E12.5)
1010  FORMAT(45X,' The Scale Factor of Abscissa: 1 Division=',E12.5)
1020  FORMAT(53X,' First Abscissa Value:       X(1)=',E12.5)
1030  FORMAT(/40X,'NOTE: In interpreting the plot, X-axis starts with'
     1 ,' X(1) value.'/45X,' Other X',1H','s can be computed ',
     2 'from X(1) and abscissa scale factor.')
1040  FORMAT(/)
1050  FORMAT('*   Values of',40X,'(Multiply by Scale Factor ',E12.5,')')
1060  FORMAT('*   Function 1',6X,11(15,5X))
1070  FORMAT(1H*,3X,'----------',9X,101A1)
1080  FORMAT('* ',E12.4,2X,'     1',101A1)
1090  FORMAT('* ',E12.4,'  X(',I3,')',A1,101A1)
1100  FORMAT('*',52X,'(Multiply by Scale Factor ',E12.5,')')
1110  FORMAT('*',19X,11(15,5X))
      END
*
*
      SUBROUTINE XYPLOT(X,Y,ND)
      REAL X(ND),Y(ND)
      INTEGER L(11),LI(101),SYMBOL
      DATA SYMBOL,JN,JP,JI,JZ/'*','-','+','1','$'/,JB/"200000000000/
      NDM=ND-1
**
**  Arrange data in ascending order of X
**
      DO 10 I=1,NDM
         IA=I+1
         DO 10 J=IA,ND
            IF(X(I).LE.X(J))GOTO 10
            TEMP=X(I); X(I)=X(J); X(J)=TEMP
            TEMP=Y(1); Y(I)=Y(J); Y(J)=TEMP
10    CONTINUE
**
**  To establish the range of Y's
**
      YMAX=-1.0E+36; YMIN=1.0E+36
      XMIN=X(1)
      DO 20 I=1,ND
         IF(Y(I).GT.YMAX) YMAX=Y(I)
         IF(Y(I).LT.YMIN) YMIN=Y(I)
20    CONTINUE
      RANGEX=X(ND)-X(1); RANGEY=YMAX-YMIN
      CALL SCALE(RANGEX,XMIN,SCLX,NSX)
      CALL SCALE(RANGEY,YMIN,SCLY,NSY)
      WRITE(6,1000)SCLX,X(1),X(ND)
      WRITE(6,1010)SCLY,YMIN,YMAX
      WRITE(6,1060)
      WRITE(6,1020)SCLY
30    DO 40 I=1 ND
         Y(I)=Y(I)/SCLY; X(I)= (X(I)-XMIN)/SCLX
40    CONTINUE
```

Listing for ENG:GRAPH.FOR

```
      NNP=100; NSX=100; LENGTH=101
      NP=100; XNP=100.0; XNS=100.0; YNS=NSY
**
**  Print ordinate scale figures
**
      DO 60 I=1,11
60    L(1)=10*I-110+NSY
      WRITE(6,1030)L
**
**  Blank out all print characters
**
      DO 70 I=1,101
70    LI(I)=JB
**
**  Start Plotting
**
      N=0;K=1
**
**  Prepare print line for the y-axis
**
80    NQ=0
      DO 90 I=1,10
         NQ=NQ+1; LI(NQ)=JP
         DO 90 J=1,9
            NQ=NQ+1
            LI(NQ)=JN
90    CONTINUE
      LI(101)=JP
      IF(N.NE.0)GOTO 110
**
**  Scale abscissa data
**
100   NX=X(K)*0.6-XNS+XNP+0.499999; NX=IABS(NX)
**
**  Check to see if data is sorted for current abscissa value
**
      IF(NX.EQ.0)GOTO 110
      IF(NX.GT.NP)NX=NP
110   IF((NX.NE.N).AND.(N.EQ.0))GOTO 130
      IF((NX.NE.N).AND.(N.NE.0))GOTO 120
**
**  Scale ordinate data
**
      KK=Y(K)+101.499999-YNS
      IF(KK.LT.1)LI(1)=JZ
      IF(KK.GT.101)LI(101)=JZ
      IF((KK.GE.1).AND.(KK.LE.101))LI(KK)=SYMBOL
      K=K+1
      IF(K.LE.ND)GOTO 100
120   DO 112 I=1,101
      IX=102-I
      IF(LI(IX).NE.JB)GOTO 114
112   CONTINUE
114   LENGTH=IX
      IF(N/6.GT.(N-1)/6)GOTO 130
**
**  Print line and data without abscissa label
**
      WRITE(6,1040)(LI(I),I=1,LENGTH)
      GOTO 140
130   NN=(N*10)/6+NSX-NNP
**
**  Print line and data with abscissa label
**
      WRITE(6,1050)NN,(LI(I),I=1,LENGTH)
140   IF(K.GE.ND)GOTO 160
      DO 150 I=1,101
150   LI(1)=JB
**
**  Set up abscissa graph lines
**
      LI(1)=JI
      IF(N/6.GT.(N-1)/6)LI(1)=JP
      GOTO 110

160   NQ=0
      DO 170 I=1,10
         NQ=NQ+1; LI(NQ)=JP
         DO 170 J=1,9
            NQ=NQ+1
            LI(NQ)=JN
170   CONTINUE
      LI(101)=JP
      WRITE(6,1040)LI
      DO 180 I=1,11
180   L(I)=10*I-110+NSY
      WRITE(6,1030)L
      WRITE(6,1020)SCLY
      WRITE(6,1060)
      WRITE(6,1060)
      RETURN
1000  FORMAT(/1H*,24X,' The Scale Factor of Abscissa: 1 Division=',
     1  E12.5/1H*,32X,' First point at ', E12.5/
     2  1H*,32X,' Last point at  ', E12.5)
1010  FORMAT(1H*/1H*,24X,' The Scale Factor of Ordinate: 1 Division=',
     1  E12.5/1H*,32X,' Range of Ordinates:   Ymin=',E12.5,
     2  /1H*,55X,'Ymax=',E12.5)
1020  FORMAT(1H*,29X,'Y-Axis (Multiply by Scale Factor',E12.5,')')
1030  FORMAT('* ',11(I4,6X))
1040  FORMAT('*    ',101A1)
1050  FORMAT(1H*,I4,101A1)
1060  FORMAT(///)
      END
*
*
      SUBROUTINE SCALE(ZRANGE,ZMIN,ZSF,NSZ)
      YPD=ZRANGE/90.; NYPD=YPD
      IF(NYPD.GT.0)GOTO 60
      DO 10 I=1,25
         YPD=10.*YPD; NYPD=YPD
         IF(NYPD.GT.0)GOTO 20
10    CONTINUE
20    RSF=10.**(-I+1); YPD=.5*YPD; NYPD=YPD
      IF(NYPD.GT.0)GOTO 40
30    ZSF=.2*RSF
      GO TO 90
40    YPD=.4*YPD; NYPD=YPD
      IF(NYPD.GT.0)GOTO 50
      ZSF=.5*RSF
      GO TO 90
50    ZSF=RSF
      GO TO 90
60    DO 70 I=2,25
         YPD=YPD/10.; NYPD=YPD
         IF(NYPD.LE.0)GOTO 80
70    CONTINUE
80    RSF=10.**(I-1); YPD=5.*YPD; NYPD=YPD
      IF(NYPD.GT.0)40,30
90    NSZ=100+10*(IFIX(ZMIN/ZSF)/10-1)
      RETURN
      END
```

The subroutine PLOT8 has the following limitations:

a. Each function must be single-valued. Therefore, this routine will not be suitable to plot a multi-valued function, such as a circle. Thus it could not be used for such applications as root locus, Nyquist plots, etc.

b. X-increments must be constant. Thus, the routine is unsuitable for data point array that does not have equally spaced x-increments.

(2) The XYPLOT Subroutine

The routine XYPLOT is designed to plot either a single-valued or a multi-valued function, with equal or unequal x-increments. It also has built-in optimal selection of scale factors. In addition, this routine is a building block in implementing a polar plot routine or a logarithmic plot routine.

The call and the parameter definitions are shown below:

```
CALL  XYPLOT(X,Y,ND)
```

where the parameters are explained below:

X,Y each a one-dimensional array X(I), Y(I) with Max I = 200. Both arrays will be altered upon return from the subroutine.

ND number of data points to be plotted.

Example: The following program shows the generation of plotting data for a parabola:

$$Y^{**}2 = 4*X + 5$$

The program (again named as SAMPLE.FOR) listing is as follows:

```
      REAL X(150),Y(150)
      DO 10 I=1,150,2
         XI=I-1;J=I+1
         X(I)=XI
         X(J)=XI
         Y(I)=SQRT(4.*XI+5.)
         Y(J)=-Y(I)
10    CONTINUE
      CALL XYPLOT(X,Y,150)
      END
```

Use the command ".EXECUTE SAMPLE,ENG:GRAPH/LIB" to execute.

The output for this example is shown in Figure 6.2. The help file and the program listings for these two programs are also included for user's reference.

There are other plotting routines in the ENG:GRAPH package. They will be only mentioned here, and further details may be found in the help-file ENG:GRAPH.HLP. The contents of ENG:GRAPH are as follows:

```
*                        The Scale Factor of Abscissa: 1 Division= 0.20000E+01
*                                First point at  0.00000E+00
*                                Last point at   0.14800E+03
*
*                        The Scale Factor of Ordinate: 1 Division= 0.10000E+01
*                                Range of Ordinates:   Ymin=-0.24434E+02
*                                                      Ymax= 0.24434E+02
*
*
*
*                                  Y-Axis (Multiply by Scale Factor 0.10000E+01)
*  -30       -20       -10         0         10        20        30        40
*   0+---------+---------+---------*-+-*---------+---------+---------+---------+----
*    I                              **        **
*    I                            **          **
*    I                           *              *
*    I                          **             **
*    I                         **               **
*   10+                       *                 *
*    I                       *                   *
*    I                      **                   **
*    I                      *                     *
*    I                     *                       *
*    I                    **                       **
*   20+                   *                         *
*    I                    *                         *
*    I                   *                           *
*    I                   *                           *
*    I                  *                             *
*    I                  *                             *
*   30+                *                               *
*    I                 *                               *
*    I                **                               **
*    I                *                                 *
*    I                *                                 *
*    I               *                                   *
*   40+              *                                   *
*    I               *                                   *
*    I              *                                     *
*    I              *                                     *
*    I             **                                     **
*    I             *                                       *
*   50+            *                                       *
*    I            **                                       **
*    I            *                                         *
*    I            *                                         *
*    I            *                                         *
*    I           *                                           *
*   60+          *                                           *
*    I           *                                           *
*    I          *                                             *
*    I          *                                             *
*    I          *                                             *
*    I         **                                             **
*   70+        *                                               *
*    I         *                                               *
*    I         *                                               *
*    +---------+---------+---------+---------+---------+---------+---------+--
*  -30       -20       -10         0         10        20        30        40
```

Figure 6.2 Output from the Subroutine XYPLOT

PLOT8(Y.NF,NP,X1,XINC)	Plot up to 8 functions.
PRINT8(Y,NF,NP,X1,XINC)	Tabulate up to 8 functions.
XYPLOT(X,Y,ND)	Plot x-y data.
XYPRNT(X,Y,ND)	Tabulation of x-y array data with x-array sorted internally in an ascending order.
SEMLOG(X,Y,ND,KODE)	Plotting with one axis on log scale. KODE=1 x-axis on log scale KODE=2 y-axis on log scale
LOGLOG(X,Y,ND)	Plotting of x-y array on log-log scales
POLAR(RHO,THETA,ND)	Polar plot for rho-theta array; theta in degrees

The users will also find that other plotting routines are available in almost every canned software package.*

6.3 Plotting on a Plotter

A plotter has a mounted pen that can be controlled for its movement as small as 1/500 inch. Therefore, it is capable to produce superb quality graphs and plots. Basically, a plotter makes a figure by moving the pen from one position on the paper to another either with the pen UP or with the pen DOWN. These movements are controlled by incorporating in the main program a series of FORTRAN subroutines, which translate the user's requirements into detailed pen movement instructions. A set of basic FORTRAN-callable subroutins will be taken up later when we get to the CalComp section of the chapter. At this point, we will look at two types of software routines that are used in producing plots quickly on the Calcomp plotter. One is a group of "quick" plot subroutines that can be incorporated into user's FORTRAN program; the other is a group of stand-alone interactive programs.*

It should be noted that the plotter is a very slow device. Therefore, plots must be queued. After the execution of the program, the software only produces a plotter-file; it does not produce a plot per se. After the plotter file is produced, it will take another monitor command "PLOT file" to actually queue a plotting job, such as:

```
.PLOT  *.PLT
```

---

*At University of Pittsburgh, in addition to the routines presented here, also available are a group of printer plot subroutines in PRG.GRAPH.REL and a stand-alone CalComp routine named PRG:GRAFIC.EXE. See References 5 and 6.

(1) The Quick Plot Subroutines

Two "quick" plot subroutines, QIKPLT AND QIKLOG, will be presented here. They are both developed at the Pitt Computer Center. The subroutine QIKPLT will plot the x-y array data on a linearly scaled plot; the subroutine QIKLOG will plot the x-y array data either on a semi-log or a log-log plot. Both subroutines were developed at the Pitt Computer Center. They produce plots that will fit into standard 8.5-by-11 inch letter-size paper. The input parameters of the subroutines, beside the required x-y array data and number of data points, will include the following:

(1) Title of the x-axis

(2) Title of the y-axis

(3) Title of the plot

(4) Option of which symbol (or no symbol) to represent every nth data point.

(5) Option of whether to join the points by lines.

(6) Option of placing grid lines (or omit them) on the plot.

The calling sequence is as follows:

```
CALL   QIKPLT(P1,P2,P3,P4,P5,P6,P7,P8,P9,P10,P11)
CALL   QIKLOG(P1,P2,P3,P4,P5,P6,P7,P8,P9,P10,P11,P12)
```

The parameters are explained as follows:

P1 X-array, real, dimensioned at least 2 more than the value of P3.

P2 Y-array, real, dimensioned at least 2 more than the value of P3.

P3 Number of data points to be plotted, integer constant/variable. If you want grid lines on the plot, specify a negative number of points. Positive P3 will omit grid lines.

P4 X-axis title. If expressed in ASCII string, it will have the form of characters enclosed within single quotes, such as 'Time in Seconds'. If expressed dimensioned variable, they have literal constant values, such as 'Time ', 'in Se', 'conds' for (KARAC(I),I=1,3). They must be in (A5) format. Maximum length of string is 45 characters or (9A5) format.

P5 Y-axis title. Same definition as in P4. Maximum is 57 characters.

P6 Title of the plot; same definition as P4. No maximum.

P7 Number of characters in x-axis title, including embedded blanks. Integer constant/variable; maximum value is 45. X-axis title is omitted if P7=0.

P8 Number of characters in y-axis title, including embedded blanks. Integer constant/variable; maximum value is 57. P8=0 means no y-axis title.

P9 Number of characters in the plot title, including embedded blanks. Integer. P9=0 means no title for the plot.

P10 Signed integer code for options of how to join points by lines:

P10 = positive integer, say n. Lines will join every nth data point by a symbol specified by the P11 parameter.

P10 = 0: The points are joined by lines with no symbols. P11 value, if given, is ignored.

P10 = negative integer, say -n. The points are marked by a symbol, specified by the parameter P11, every nth data point. No line is drawn to join them.

P11 Integer code for the symbol choice to mark a data point:

P11 Code	Symbol	P11 Cod	Symbol
0	□	8	Z
1	○	9	Y
2	△	10	⊠
3	+	11	*
4	×	12	⧖
5	◇	13	I
6	↑	14	✲
7	⧗		

P12 Option code for log plot routine:

P12=-1 to request a semi-log plot, with x on log scale
P12= 0 to request a log-log plot
P12=+1 to request a semi-log plot, with y on log scale.

The process of using these quick subroutines is quite straight forward: In the main program, be sure to dimension X and Y arrays at a dimension two more than actually needed by the array. Also, dimension those ASCII variables that are needed for the titles. Then:

(1) Read, or generate, or calculate the x-y arrays.

(2) Call the quick plot routine

Example:

```
 CALL QIKPLT(X,Y,-50,'Time in Seconds','Voltage in Volts',
1 'FILTER RESPONSE',15,16,15,-1,4)
```

This subprogram call will produce a plot of 50 data points not joined by lines, but each point is marked by a X symbol. The titles are supplied as indicated in the parameter list.

Example:

Here we will recount how the plots in Figure 5.4 were made (see Page 220). The data from the CSMP run were saved as three files DATA1.DAT, DATA2.DAT and DATA3.DAT and in a a format of (2E).

The following program was prepared:

```
      REAL X(100),Y(100)
      DOUBLE PRECISION FLNAME(3)
      DATA FLNAME/'SUSP.DAT','IMMP.DAT','INFP.DAT'/
      DO 500 K=1,3
        OPEN(UNIT=1,FILE=FLNAME(K))
        DO 10 I=1,1000
          READ(1,50,END=20)X(I),Y(I)
10      CONTINUE
        STOP
20      NPT=I-1
*
        IF(K.EQ.1)CALL QIKPLT(X,Y,NPT,
     1  'Time in Seconds','Voltage in Volts',
     2  'FILTER INPUT VOLTAGE',15,16,20,0,0)
        IF(K.GE.2CALL QIKPLT(X,Y,NPT,
     1  'Time in Seconds','Voltage in Volts',
     2  'FILTER OUTPUT TRANSIENT',15,16,22,0,0)
        CLOSE(UNIT=1)
*
500   CONTINUE
50    FORMAT(1X,2F15.4)
      STOP
      END
```

The output from this program has been previously shown in the last chapter as Figure 5.4(a).

(2) The Interactive Plot Programs

Although the quick plot routines are very convenient to use, they both have long lists of parameters, and they tend to be error-prone. Therefore, convenience can be enhanced by incorporating the graphic routines into an interactive program, in which the plot routine parameters will be entered and guided by an interactive dialogue.

Two programs using this approach will be presented here:

The CALPLT Program in ENG:

One of the program unit in ENG: is CALPLT, an interactive program for CalComp plotting. By interactive dialogue, the user enters inputs, such as the data filename, and all titles. The counting of characters is done automatically within the program. The listing of the program is shown. To invoke and execute the program, use the standard Engineering Library call:"PIL ENG:CALPLT". For other Library programs, see Appendix B.

```
* ********************************
*                                *
*          ENG:CALPLT.FOR        *
*                                *
* ********************************
*   Interactive program to do a Calcomp plotter job
*   Require data set file with one point per line (2 real constants
*     in either (2E) or (2F) format.
*   Will produce as an output a plotter file XXXXXX.PLT ready to queue.
*   Option of linear, semi-log, or log-log plot
*   Maximum 200 data points
      REAL X(202),Y(202)
      DOUBLE PRECISION FLNAME
      INTEGER XTITLE(12),YTITLE(12),GTITLE(12)
      DATA XTITLE/12*'     '/,YTITLE/12*'     '/,GTITLE/12*'     '/
      DATA KARE,KARF/'E','F'/
      WRITE(6,80); READ(5,90)KOP
      WRITE(6,100); READ(5,110)FLNAME
      OPEN(UNIT=1,FILE=FLNAME)
      WRITE(6,120); READ(5,130)KKK
      DO 10 I=1,1000
        IF(KKK.EQ.KARE)READ(1,140,END=20)X(I),Y(I)
        IF(KKK.EQ.KARF)READ(1,150,END=20)X(I),Y(I)
10    CONTINUE
      STOP
20    NPT=I-1
30    WRITE(6,160); READ(5,130)(XTITLE(I),I=1,12)
      CALL KARAC(XTITLE,LX);IF(LX.GT.45)WRITE(6,170);IF(LX.GT.45)GOTO 30
40    WRITE(6,180); READ(5,130)(YTITLE(I),I=1,12)
      CALL KARAC(YTITLE,LY);IF(LY.GT.57)WRITE(6,170);IF(LY.GT.57)GOTO 40
      WRITE(6,190); READ(5,130)(GTITLE(I),I=1,12)
      CALL KARAC(GTITLE,LG)
      IF(KOP.EQ.1)CALL QIKPLT(X,Y,NPT,XTITLE,YTITLE,GTITLE,
     1   LX,LY,LG,0,0)
      IF(KOP.EQ.2)LOGT=-1
      IF(KOP.EQ.3)LOGT=1
      IF(KOP.EQ.4)LOGT=0
      IF(KOP.GT.1)CALL QIKLOG(X,Y,NPT,XTITLE,YTITLE,GTITLE,
     1   LX,LY,LG,0,0,LOGT)
80    FORMAT(/' PLOT OPTIONS:   OPTION = 1 FOR LINEAR PLOT',
     1 /17X,'OPTION = 2 FOR SEMI-LOG PLOT, X-AXIS LOG SCALE',
      2 /17X,'OPTION = 3 FOR SEMI-LOG PLOT, Y-AXIS LOG SCALE',
      3 /17X,'OPTION = 4 FOR LOG-LOG PLOT.'//
      4 ' ENTER OPTION = '$)
90    FORMAT(I)
100   FORMAT(/' INPUT FILENAME='$)
110   FORMAT(A10)
120   FORMAT(/' DATA FORMAT IN EITHER 2E OR 2F FORMAT YOUR ',
     1  'FILE, E OR F?'$)
130   FORMAT(12A5)
140   FORMAT(2E)
150   FORMAT(2F)
160   FORMAT(/' X-AXIS TITLE='$)
170   FORMAT(/' TITLE TOO LONG, TRY AGAIN.')
180   FORMAT(/' Y-AXIS TITLE='$)
190   FORMAT(/' PLOT TITLE='$)
      END
************************
*  SUBROUTINE   KARAC  *
************************
*   To count number of character in an ASCII variable array
      SUBROUTINE KARAC(ASCII,LENGTH)
      INTEGER ASCII(12)
      DATA JB/32/
      DO 10 I=1,12
        IX=13-I
        DO 10 J=1,5
          JX=6-J; JBIT=(JX-1)*7; JKAR=LDB(JBIT,7,ASCII(IX))
          IF(JKAR.NE.JB)GOTO 20
10    CONTINUE
      LENGTH=0; RETURN
20    LENGTH=5*(IX-1)+JX; RETURN
      END
```

Listing for ENG:CALPLT.FOR

The PLOTIT Program in USL:

An excellent and convenient interactive program with many plot options is available in the User Library USL: It was developed at the University of Pittsburgh by Professor Frederick Gottlieb of the Biological Sciences Department. It has options of entering the data by stored files, or by typing in the data at the terminal. It allows the user to choose, among others, number of curves on one plot (6 maximum), x- or y-axis as the "long" axis or else a square plot, labels and titles, tic marks, data point symbols, smooth (French curve fit) or connecting-line curves. Perhaps, the best way to show how this interactive program works is to reproduce the interactive dialogue of an actual run.

Therefore, the data files DA.DAT, DB.DAT, and DC.DAT that produce Figure 6.1 are used again for illustration. As in the consistent practice in this book, the user's response in a dialogue is re-typed with italics and underscored.

6.4 Preview of Plotter Output

The plotter is a very slow device, and therefore when a plotter job is queued, there is generally a long turn-around time. If a mistake is made, or if subsequent processing is dependent on the graphic output result, the delay here represents a serious bottleneck. Typing or printing out a plot file will be useless, because it contains codes that can be understood only by the plotter.

At the University of Pittsburgh, both the CalComp Plotter and the Tektronix 4010-series graphics terminals are available and supported. A program TEKPLT has been implemented by the Computer Center staff to display a plotter file on the Tektronix graphic terminal, thus providing an opportunity to preview the plotter output. TEKPLT may be called and executed by a monitor command of:

```
.R TEKPLT
```

The system will respond with a request for the plotter file name:

```
Enter file name >
```

If you don't know the name of the plot file, use a DIRECTORY command to find out, for example:

```
.DIRECT *.PLT
```

The filenames of all plot files will be printed out.

After the filename of the plotter file is supplied, TEKPLT will draw the plot on the Tektronix screen, and will automatically adjust the size to fit the screen. If the long and narrow dimension of the plot fits the wrong way, give a TEKPLT command of "I" (without carriage return) to rotate the plot 90 degrees. Other TEKPLT commands are detailed in References 7 and 8.

```
.RUN USL:PLOTIT

Welcome to PLOTIT (Rev. 9 Jan 80.)
This program will call existing data sets and plot them with up to
6 curves per plot (i.e., axis set), and upper/lower case notation.
It is suggested that you use JOTTER to write the data sets for plotting.

Do you want the FORMAT details for data sets which are compatible
with PLOTIT? (Answer yes or no.)>YES

The data set lines must be in the format (1x,3f15.__), where the first two value
s are X and Y coordinate values
and the third value is an error bar value - usually the standard error of the me
an of Y- and will have the value of
zero if no error bars are to be drawn.
 Data sets should not exceed 200 X, Y and Error values.

Do you want to write a data set?(answer yes or no)>NO

Do you want to plot now? (Answer yes or no)>YES

Please give me a name for this plot file.
(6 character maximum)>SAMPLE

How many curves on this plot? (answer with an integer between 1 and 6)>3

What is the name and extension of the first data file? (name.ext)>DA.DAT

What is the name and extension of the second data file?(name.ext)>DB.DAT

What is the name and extension of the third data file? (name.ext)>DC.DAT

Which is the long axis? (Type x or y or s [for a square box plot])>Y

Want plot with numbers and labels?>YES

Is either axis a log axis? (answer yes or no)>NO

What is the value at the x-origin?>0

What is the highest axis value of x?>50

How many scale units between labeled tics on x?>10

Want tics between labeled x tics?(answer yes or no)>YES

How many tics between labeled tics?(integer between' 1 and 9)>9

What is the value at the y-origin?>0

What is the highest axis value of y?>1000

How many scale units between labeled tics on y?>100

Want tics between labeled y tics? (answer yes or no)>YES

How many tics between labeled tics?(integer between' 1 and 9)>9

Type the x-axis label (30 character maximum)
Label will be in upper and lower case, the symbol "~" will plot as "±",
 enter the label with a "@" after the last character
:<--------30 characters------->:(ie, between the two ":")
>NUMBER OF DAYS@

Type the y-axis label(30 character maximum)
Label will be in upper and lower case, the symbol "~" will' plot as "±",
enter the label with a "@" after the last character
:<--------30 characters------->:(ie,between the two ":")
>NUMBER OF PERSONS@

How many decimal places in the x tags? (0,1,2,etc.   or 9=integer)>9

How many decimal places in the y tags? (0,1,2,etc.  or 9=integer)>9

Do you want to alter the size of the output plot?  Answer yes or no.>YES

You may shrink the plot to as small at 0.25 x; or you may expand up to 2.00x.
 The 8 1/2 x 11 inch page- 'box will be omitted.
 Please choose a magnification  factor between 0.25 and 2.00>0.6

Do you want a ticked upper and right axis also? Answer yes or no.>YES


First Curve

How many data points to be ploted? (enter an integer between 1 and 200)>40

Want symbols at data points?(answer yes or no)>YES

Please choose your symbol for this plot.
type the appropriate two letter code from this list:
 Code Letters     Symbol
      oc        =  open circle
      fc        =  filled circle
      od        =  open diamond
      fd        =  filled diamond
      os        =  open square
      fs        =  filled square
      ot        =  open triangle
      ft        =  filled triangle
      cr        =  cross (x)
Which symbol do you want?>FC

Do you want a smoothed french curve fit?(answer yes or no)>FD

Answer yes or no!>YES
```

```
Second Curve

How many data points to be ploted? (enter an integer between 1 and 200)>46

Want symbols at data points?(answer yes or no)>YES

Which symbol do you want?>PD

Do you want a smoothed french curve fit?(answer yes or no)>YES

Third Curve

How many data points to be ploted? (enter an integer between 1 and 200)>46

Want symbols at data points?(answer yes or no)>YES

Which symbol do you want?>FT

Do you want a smoothed french curve fit?(answer yes or no)>YES

The plot file SAMPLE.plt is now in your directory.

Want to draw another graph now? answer yes or no>NO

Then we'll stop now, Have a good day.
STOP

End of execution    FOROTS 5B(1001)
CPU time: 4.70  Elapsed time: 7:25.28
EXIT

.DIRECT *.PLT

SAMPLE  PLT    48  <057>   29-Oct-80    USRB:   [115036,320571]

.PLOT SAMPLE.PLT
Total of 2 minutes in 1 file in PLT request / Sequence number 10045
```

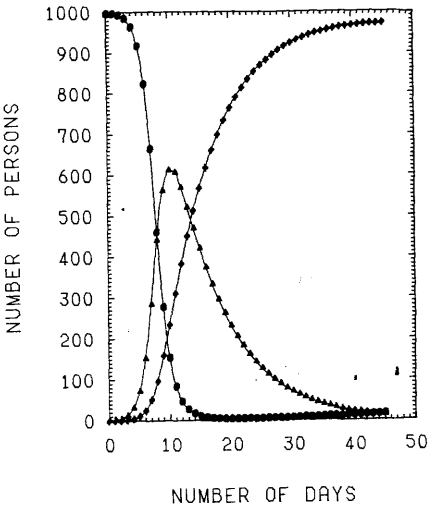
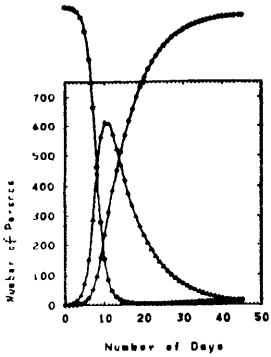


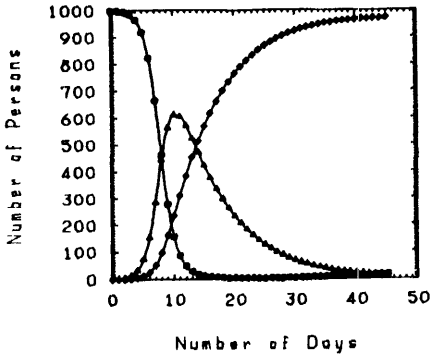
Figure 6.3 Plot Produced by PLOTIT

Figure 6.4 includes a group of reproductions of hardcopies of the TEKTRONIX-4010 displays for the PLOTIT runs that prepared Figure 6.3. During the run, some mistakes were inadvertently made in the input phase of PLOTIT. With the preview capability of the TEKPLT program, these mistakes were evident in the preview displays. Valuable time and plotter resources were saved. After a preview session, only the satisfactory plot file was submitted for the actual plotter job.

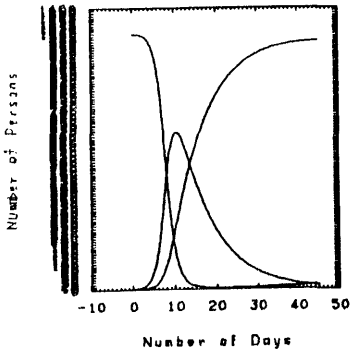
Some DEC-10 installations have developed simulator programs that will print out a CalComp plot on a printer for preview purpose. Users should check with their local installation about their availability.



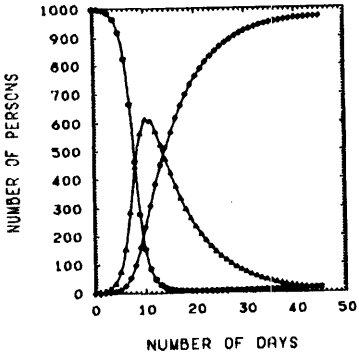
Y-axis scale error



Y-axis too short



Labeling all messed up



Chosen for Figure 6.3

Figure 6.4 Plots Previewed Using TEKPLT Program

GENERAL GRAPHICS

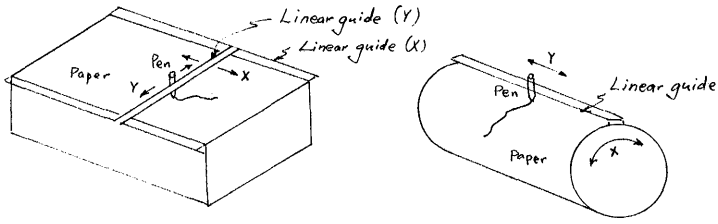
In the general graphics applications using a plotter or a graphic terminal, software becomes very much hardware-dependent. The result is that if two DEC-10 facilities have different graphic hardwares, their software may not necessarily be compatible. Two types of graphics hardware will be considered here: one is to produce hardcopy and is essentially a plotter; and the other is to produce soft copy and is essentially a CRT terminal. Selected for inclusion in this part of the chapter are the discussions on two software packages: the CalComp subprograms, and the Tektronix PLOT-10 System. Both are in the forms of FORTRAN-callable subroutines.

A PRIMER FOR THE CALCOMP PLOTTER

6.5 Basic Principle of a Digital Plotter

There are many different manufacturers of digital plotters. Probably the most commonly used type is the CalComp plotter, manufactured by California Computer Products, Inc., Anaheim, California. Actually, the basic principle for all digital plotters is very similar.

In a digital plotter, the pen is attached to a ribbon or wire and can move along a linear guide. This is the y-direction movement. To the right angle of the pen guide, the x-axis movement is provided by either of two ways as shown in Figure 6.5.



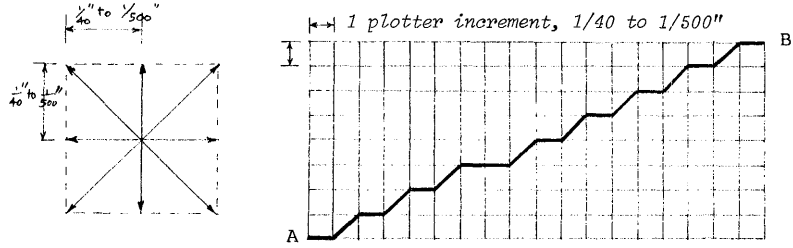
Flatbed Type Plotter Drum Type Plotter

Figure 6.5 Two Basic Types of Digital Plotters

In a flat-bed type plotter, the movement of another linear guide perpendicular to the y-movement provides the x-movement. In a drum-type plotter, the pen can only travel in the y-direction, but the drum may turn. Its rotation, or the relative motion with respectt to the pen, provides the x-movement.

Movements in each direction are controlled by precision servo step-motors, and each "step" movement is translated by a reduction gear train to a precise pen movement in the x- or y-direction. This incremental pen movement defines the basic resolution of a plotter. Commercial incremental digital plotter ranges from 1/40 inch to 1/500 inch or better.

Since the x- and y-movements are incremental, each step of pen movement is a combination of the x-y increments. As a result, there are only eight possible ways that a pen can move in one increment.



(a) Basic Incremental Pen Movements

b) Drawing a "Straight" Line (Enlarged approximately 75 times)

Figure 6.6 Incremental Pen Movements

As shown in Figure 6.6(a), the eight possible ways of incremental pen movement, borrowing the term from the points of a compass, are E, NE, N, NW, W, SW, S, and SE. Thus, to move linearly from point A to point B, the actual path of the pen, and therefore the trace it draws, is actually a zigzag step line as shown in Figure 6.6.(b). Fortunately, the resolution of the plotter is fine enough that the zigzags will not be visually noticeable.

In addition to the x- and y-movements, the pen can be lifted or lowered on paper. When the pen is lowered and moved, it traces a line. When the pen is lifted and moved, there is no line. This simple pen ability is necessary in order to skip from one point to another.

Therefore, a basic plotter instruction is rather primitive. It tells the plotter to lift or lower the pen, or keeps the pen position as is. It tells the plotter one of the eight directions of incremental pen motion. Such raw instructions are too cumbersome for application. Therefore, a plotter manufacture usually provides a basic plotter language that includes a set of plotter instructions. Thus, CalComp has CalComp language; Gould has Gould language; Hewlett-Packard has its language. The list goes on. To compound the proliferation, each local installation then develops its own high-level (FORTRAN-callable) subroutines based on the plotter language. Hence, the subroutines used at one installation may not be run at another. When a user faces this situation, there are two options. One way is to obtain a software package from the installation, where the language was developed, by purchase, lease, or exchange. The other way is to develop a simulator which translates the foreign subroutine calls into the local subroutine calls.

Next, let us plan a drawing as shown in Figure 6.7. To obtain a perspective, the figure is superimposed on a coordiante grid lines, with the lower left corner being the origin (0,0).

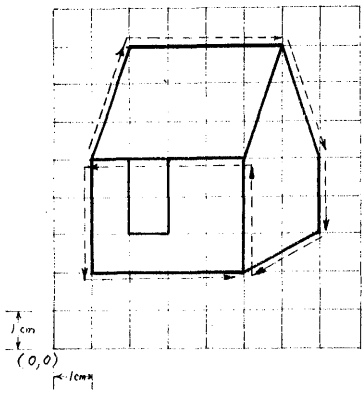


Figure 6.7 Drawing a Figure

Pen positions and movements can be planned to draw the figure:

Pen Position (UP or DOWN)	Pen Movement Destination
up	(1,5)
down	(2,8)
down	(6,8)
down	(7,5)
down	(7,3)
down	(5,2)
down	(5,5)
down	(1,5)
down	(1,2)
down	(5,2)
up	(5,5)
down	(6,8)
up	(3,5)
down	(3,3)
down	(2,3)
down	(2,5)
STOP	

In the presentation that follows, the materials will be based on the set of plotter routines that were either furnished by CalComp or developed at the University of Pittsburgh.

6.6 A Primer on CalComp Plotter Subroutines

At the University of Pittsburgh, a CalComp 936 metric digital pen plotter is installed with the DEC System-10. A number of the plotter routines are supplied by California Computer Products, Inc., the manufacturer of the plotter, and many were developed by the Computer Center staff. Collectively, these FORTRAN-callable subroutines are stored as PRG:PLTLIB.REL, which is also incorporated into the FORTRAN system library. Thus when a program PRGM.FOR containing plotter routines is to be executed, the monitor command issued should be:

```
                    .EXECUTE PRGM.FOR
or,                 .EXECUTE PRGM.FOR, PRG:PLTLIB/LIB
```

When the execution is completed, there is a plot-file (with an extension PLT) generated in the user's disk. A PLOT monitor command for that plot file will queue the plot-job, and a plotter output will be made.

Among the CalComp subroutines, many parameters pertain to the lengths and coordinates. It may be set to either the English system (in inches) or the Metric system (in centimeters) by calling the subroutine METRIC:

```
CALL METRIC(LOGIC)
```

where: LOGIC=.TRUE. for linear measurements in centimeters.
LOGIC=.FALSE. for linear measurements in inches.

This subroutine may be placed anywhere in a program, and the system of measurement selection takes effect after that subroutine call. If no such subroutine is called, the default system is Metric.

Thus, one can not only select the system but also switch the system from one to another at will. In the presentation of the Plotter Primer of Subroutines that follows, all linear measurements will be in centimeters for reason of consistency. If a user wishes to use these subroutines in English system (in inches), a "CALL METRIC(.FALSE.)" FORTRAN statement should be given first.

The selected CalComp subroutines are divided into six groups:

(1) Initializing and terminating a plot
(2) Re-defining the new origin and scales
(3) Basic pen movements
(4) Annotation of symbols and numbers
(5) Axis and scales
(6) Lines and Curves

They are now presented next.

(1) Initializing and terminating a plot

When a plot job is initialized, three major events will be performed: Plot area bounds will be established; a plot file is opened in your disk; and all options are assigned with default values.

When a plot job is finished, that plot-file must be closed. Even within the same program, one should always terminate (close) one plot-file before enbarking on another plot job. Failing to that would get a drawing with several drawings superimposed on each other.

The subroutines of this group are shown in Table 6.1.

(2) Redefining the origin and the scales

When a plot job is initialized, the origin is set at the lower left corner of the plot area, and scale is set to the actual centimeters or inches. This may be very inconvenient. Allowing redefining the new origin and new scales are a group of subroutines as shown in Table 6.2.

(3) Basic pen movements

This is perhaps the most fundamental plotter routine. It moves the pen from the current position to another specified position, leaving on the paper with or without a tracing. All plotting routines in the CalComp software are built upon this routine. This is shown in Table 6.3.

(4) Annotation of symbols and numbers

Annotations in text are represented in ASCII string constant or in string array variable with A5 formats. Symbols are defined according to a coding table, where each symbol is represented by the decimal equivalent of its ASCII code. The codes for symbols are shown in Table 6.4. The subroutines for annotation are shown in Table 6.5.

(5) Axis and scales

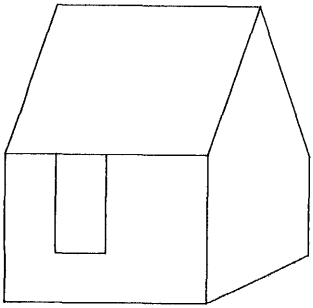
Among the most useful routines are those which automatically scales and draws the axis, either linear type or logarithmic type. The axis can be labeled with symbols and numbers and tic marks. The group of routines that draw axis and scale them are shown in Table 6.6.

(6) Lines and curves

In this group are routines that plot lines in cartesian or polar coordinates, in linear or logarithmic scales. Option of smoothing is available. Higher order routines of drawing common geometric patterns, such as circles, ellipses and polygons are provided. These routines are grouped in Table 6.7 and Table 6.8.

6.7 Examples of CalComp Programming

Example 1: Write a program to draw the figure as shown in Figure 6.7.



FORTRAN program:

```
CALL GRAPH(10.0,10.)
CALL PLOT(1.0,5.0,3)
CALL PLOT(2.0,8.0,2)
CALL PLOT(6.0,8.0,2)
CALL PLOT(7.0,5.0,2)
CALL PLOT(7.0,3.0,2)
CALL PLOT(5.0,2.0,2)
CALL PLOT(5.0,5.0,2)
CALL PLOT(1.0,5.0,2)
CALL PLOT(1.0,2.0,2)
CALL PLOT(5.0,2.0,2)
CALL PLOT(5.0,5.0,3)
CALL PLOT(6.0,8.0,2)
CALL PLOT(3.0,5.0,3)
CALL PLOT(3.0,3.0,2)
CALL PLOT(2.0,3.0,2)
CALL PLOT(2.0,5.0,2)
CALL ENDPAG
END
```

After the execution of the program, find out what is the computer-assigned name of the plot-file by a command ".DIRECTORY *.PLT". Suppose the directory shows a "QRUS3.PLT" file created. Then submit a plotter queue job by another command ".PLOT QRUS3.PLT".

The series of monitor command given to go through the sequence was:

```
.EXECUTE PRGM,PRG:PLTLIB/LIB
.PLOT *.PLT
```

Subroutine:	*CALL GRAPH(WIDTH, HEIGHT)*
Function:	To initialize a plot by: setting the plot area limit, opening an output plot-file in the user's disk, and moving the pen (UP) to the lower left corner of the plot area (coordinate 0,0). This call must precede all other plotter subroutines.
Parameters:	
WIDTH	Width of plot area permitted, ranging from 1 to 130 centimeters.
HEIGHT	Height of plotting area permitted, ranging from 1 to 80 centimeters.

Subroutine:	*CALL ENDPAG*
Function:	To terminate the plot and to close the plot-file. This call must be given at the end of a plot; otherwise, no plot-file will be stored.
Parameters:	None

Subroutine:	*CALL GRAPH2*
Function:	To re-initialize another plot in the same program. The plot size remains the same as defined by a previous GRAPH subroutine call. If there is no previous GRAPH call, the size is set to a default size of 30.5cm(width) by 23cm(height).
Parameters:	None

Table 6.1 CalComp Subroutines - Initializing and Terminating a Plot

Subroutine:	*CALL ORIGIN(XNEW, YNEW)*
Function:	To translate the origin to a new point with the coordinates (XNEW,YNEW) with respect to the lower left corner of the plot area defined by the subroutine GRAPH call. When the plot is first initialized, the origin is set to that point.
Parameters:	
XNEW,YNEW	The coordinates of the new origin with respect to the lower left corner of the plot as defined by GRAPH.

Subroutine:	*CALL PSCALE(XCALE, YSCALE)*
Function:	To change the size of the plot.
Parameters:	
XSCALE,YSCALE	The x- and y-scale factors respectively applied to the abscissa and ordinate values.

Subroutine:	*CALL LORGIN(XLORGN, YLORGN)*
Function:	To specify a logical origin if the size of a plot is changed by the subroutine PSCALE call.
Parameters:	
XLORGN,YLORGN	The logical origin with coordinates with reference to the previously defined ORIGIN.

Table 6.2 CalComp Subroutines - Redefining Origin and Scales

Subroutine: *CALL PLOT(XEND, YEND, KODE)*

Function: To move the pen from the present position to (XEND,YEND) with the pen UP/DOWN position defined by KODE.

Parameters:

XEND,YEND	The coordinates of the destination point.
KODE	Pen code: 3 for pen UP. 2 for pen DOWN 0,1 for no change

This is the basic subroutine upon which all other plotting routines are based on.

Subroutine: *CALL PENDN*

Subroutine: *CALL PENUP*

Function: To lower or to lift the pen, no x-y movement.

Parameters: None

Subroutine: *CALL SMOOT(XEND, YEND, KODE)*

Function: To plot a smooth curve using a spline-fit technique.

Parameters:

XEND,YEND	As defined in PLOT or as below.
KODE	Initial call of SMOOT must have KODE=0 or -1 to place it in the "smoothing mode."

KODE	Meaning
0	Define (XEND,YEND) as an initial point. The last point and the intial point will not be joined later. This is to draw an open curve.
-1	Define (XEND,YEND) as an initial point on the curve. The last and the first point will be joined later. This is to draw a closed curve.

After the initialization of smoothing mode, other values of KODE may be applied and interpreted as:

KODE	Meaning
-2	Plot smoothed line, pen DOWN.
-3	Move the pen along the smoothed line, pen UP.
+2,+3	Interpreted the same way as in PLOT subroutine.

Subroutine: *CALL DASHP(XEND, YEND, DASH)*

Function: To draw a dash line from the current position to (XEND,YEND).

Parameters:

XEND,YEND	The coordinates of the destination point
DASH	The length of each dash.

Table 6.3 CalComp Subroutines - Basic Pen Movement

0	□	20	≡	40	(	60	<	80	P	100	D	120	X		
1	○	21	∧	41	)	61	=	81	Q	101	E	121	Y		
2	△	22	∨	42	*	62	>	82	R	102	F	122	Z		
3	⊣	23	↓	43	+	63	?	83	S	103	G	123	[		
4	×	24	Δ	44	,	64	@	84	T	104	H	124	\		
5	◇	25	≠	45	-	65	A	85	U	105	I	125	]		
6	♠	26		46	.	66	B	86	V	106	J	126	↑		
7	✕	27		47	/	67	C	87	W	107	K	127	←		
8	z	28		48	0	68	D	88	X	108	L				
9	Y	29		49	1	69	E	89	Y	109	M				
10	✖	30		50	2	70	F	90	Z	110	N				
11	✱	31		51	3	71	G	91	[	111	O				
12	✖	32		52	4	72	H	92	\	112	P				
13	ı	33	!	53	5	73	I	93	]	113	Q				
14	✱	34	"	54	6	74	J	94	↑	114	R				
15	-	35	#	55	7	75	K	95	←	115	S				
16	ı	36	$	56	8	76	L	96	@	116	T				
17	Σ	37	%	57	9	77	M	97	A	117	U				
18	≥	38	&	58	:	78	N	98	B	118	V				
19	≤	39	'	59	;	79	O	99	C	119	W				

Table 6.4 CalComp Symbol Table

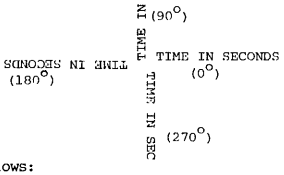
Omitting those code numbers from 0 to 31, this code table is also known as the ADE (ASCII DECIMAL EQUIVALENT) code.

Subroutine: *CALL SYMBOL(XBEGIN, YBEGIN, HEIGHT, KARAC, ANGLE, KODE)*

Function: To draw the title text or a symbol.

Parameters:

XBEGIN,YBEGIN The coordinates of the lower left corner of the first character to be drawn.
HEIGHT The height of each character or symbol.
ANGLE Angle in degrees at which the characters are drawn. For example:



KODE,HEIGHT Defined as follows:

KODE	Meaning
+n	KODE = number of characters in the KARAC parameter KARAC = Character string to be drawn; in ASCII constant (enclosed in single quotes) or as ASCII array variable. Array variable must be dimensioned in the calling program and in A5 format.
0,-n	KODE = -2: Center the symbol, Pen DOWN. KODE = -1: Center the symbol, Pen UP. KODE = 0: Symbol not centered, Pen UP. KARAC = an integer code of plotter symbol as defined in Table 6.4.

Subroutine: *CALL NUMBER(XBEGIN, YBEGIN, HEIGHT, FLOAT, ANGLE, KODE)*

Function: To draw a number in FORTRAN F-format (floating point) format.

Parameters:

XBEGIN,YBEGIN,HEIGHT,ANGLE As defined in the subroutine SYMBOL.
FLOAT Real constant/variable for the floating point number to be drawn.
KODE As defined below:

KODE	Meaning
+n	Number of digits to the right of decimal point.
0	Rounded integer, drawn with a decimal point.
-1	Rounded integer, drawn without a decimal point.
-n	P-Format scaling by the formula: FLOAT*(10**(KODE+1)). The constant is then rounded and drawn without a decimal point.

Subroutine: *CALL PLTSYM(XBEGIN, YBEGIN, HEIGHT, KARAC, ANGLE, NCHAR, NAME)*

Function: To plot symbols and characters whose positions are controlled by a user-supplied subprogram.

Parameters:

XBEGIN,YBEGIN,HEIGHT,KARAC,ANGLE As defined in the subroutine SYMBOL.
NCHAR Number of characters.
NAME Name of the user-supplied subprogram.

Table 6.5 CalComp Subroutines - Annotations with Symbols and Numbers

Subroutine:	*CALL SCALE(ARRAY, AXLEN, NPT, INC)*
Subroutine:	*CALL SCALG(ARRAY, AXLEN, NPT, INC)*
Function:	To determine the scale factor, number of data units per unit length (cm or inch) along the specified axis. SCALE:-----linear scale SCALG:-----Log scale Use this subroutine to scale separately the x-array and the y-array.
Parameters:	
ARRAY	The first array element of the data point array. The arrray must be dimensioned in the calling program at least (NPT+2). On return from this subroutine (when INC=1): ARRAY(NPT+1) = First value = FIRSTV used in subroutine AXIS or LGAXS ARRAY(NPT+2) = scale factor = DELTAV used in subroutine AXIS or LGAXS
AXLEN	Length of the axis
NPT	Number of data values contained in the array ARRAY.
INC	Increment with which data values are selected. INC=2 means every other point.

Subroutine:	*CALL AXIS(XBEGIN,YBEGIN,KARAC,NCHAR,AXLEN,ANGLE,FIRSTV,DELTAV)*
Subroutine:	*CALL LGAXS(XBEGIN,YBEGIN,KARAC,NCHAR,AXLEN,ANGLE,FIRSTV,DELTAV)*
Function:	To plot a linear axis (AXIS), or log axis (LGAXS) and annotate it with labels, title and tic marks.
Parameters:	
XBEGIN,YBEGIN,KARAC	As defined in SYMBOL subroutine.
NCHAR	Signed number of characters in KARAC. If there is no title, KARAC=' ', AND NCHAR=1. If NCHAR=positive, the annotation is placed on the counter-clockwise side of the axis. (for Y-axis title) If NCHAR=negative, the annotation is placed on the clockwise side of the axis. (for X-axis title)
AXLEN	As defined in subroutine SCALE.
ANGLE	As defined in subroutine SYMBOL.
FIRSTV	First value drawn at the first tic mark, obtained from SCALE subroutien.
DELTAV	Number of data units per unit length, obtained from SCALE subroutine.

Subroutine:	*CALL GRID(XBEGIN, YBEGIN, DX, DY, NBLKX, NBLKY)*
Function:	To draw a linear and rectangular grid.
Parameters:	
XBEGIN,YBEGIN	The lower left corner coordinates of the grid.
DX	Displacement between grid lines along the x-axis.
DY	Displacement between drig lines along the y-axis.
NBLKX	Number of blocks in the completed grid in the x-direction.
NBLKY	Number of blocks in the completed grid in the y-direction.

Table 6.6 CalComp Subroutines - Axis, Scales and Labels

Subroutine:	*CALL LINE(XARRAY,YARRAY,NPT,INC,LINCOD,SYMBOL)*
Subroutine:	*CALL DASHL(XARRAY,YARRAY,NPT,INC)*
Subroutine:	*CALL LGLIN(XARRAY,YARRAY,NPT,INC,LINCOD,SYMBOL,KODE)*
Subroutine:	*CALL POLAR(RARRAY,TARRAY,NPT,INC,LINCOD,SYMBOL,RMAX,DR)*
Function:	To draw a curve through the data points. LINE:------To plot a linear plot with solid line. DASHL:-----To plot a linear plot with dash line. LGLIN:-----To plot a semi-log or log-log plot. POLAR:-----To plot a polar plot.
Parameters:	
XARRAY,YARRAY	Data point arrays in Cartesian coordinates.
RARRAT,TARRAY	Data point arrays in Polar coordinates.
NPT	Number of data points to be plotted.
INC	Increment between data points.
LINCOD	Line code as defined below:

LINCOD	Meaning
0	Line plotted but no symbol.
1	Line and symbols at every data point.
2	Line and symbols at every other point.
n	Line and symbols at every nth point.
-n	Symbols at every nth point, no line.

SYMBOL	An integer value specifying the symbol as defined in Table 6.4.
KODE	Code for log plot type:

KODE	Meaning
-1	Semi-log plot, X in log scale.
0	Log-log plot
1	Semi-log plot, Y in log scale.

RMAX	Maximum radius in actual centimeters or inches needed for the plot. If RMAX is zero or negative, the parameter DR is used as a scale factor.
DR	Scale factor for the plot. If RMAX is positive, DR is computed by the subroutine POLAR. If RMAX is zero or negative, the user must supply a computed DR.

Table 6.7 CalComp Subroutines - Lines and Curves Plotting

Example 2: Either of two following programs will produce a grid as shown below:

```
      CALL GRAPH(12.0,12.0)
      CALL ORIGIN(1.0,1.0)
      Y=-0.1
      DO 10 I=1,101
         Y=Y+0.1; I10=MOD(I-1,10)
         CALL PLOT(10.0,Y,2)
         IF(I10.NE.0)GOTO 10
         CALL PLOT(10.03,Y+.03,3)
         CALL PLOT(0.03,Y+.03,2)
10    CONTINUE
      X=-0.1
      DO 20 I=1,101
         X=X+0.1; I10=MOD(I-1,10)
         CALL PLOT(X,0.0,3)
         CALL PLOT(X,10.0,2)
         IF(I10.NE.10)GOTO 20
         CALL PLOT(X+.03,10.0,3)
         CALL PLOT(X+.03,0.0,2)
20    CONTINUE
      CALL ENDPAG
      END
```

```
CALL GRAPH(12.0,12.0)
CALL GRID(0.1,0.1,1.0,1.0,100,100)
CALL GRID(1.03,1.03,1.0,1.0,10,10)
CALL ENDPAG
END
```

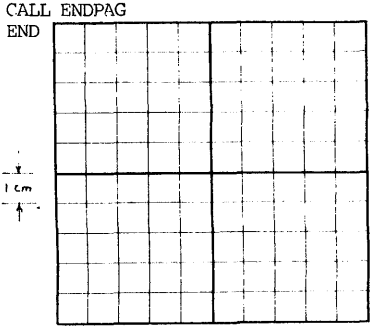


Figure 6.9 Plotting a Grid

Subroutine: *CALL CIRCL(XBEGIN, YBEGIN, THO, THF, RO, RF, DI)*

Function: To plot a circular arc.

Parameters:

XBEGIN,YBEGIN	The coordinates of the starting point of the circular arc.
THO	Radial angel at the start of the arc, in degrees
THF	Radial angle at the end of the arc, in degrees
RO	Starting radius of the arc
RF	Ending radius of the arc.
DI	DI=0.0 for solid line; DI=0.5 for dashed line.

Subroutine: *CALL ELIPS(XBEGIN, YBEGIN, RMAJ, RMIN, ANGLE, THO, THF, KODE)*

Function: To plot an elliptical arc.

Parameters:

XBEGIN,YBEGIN	The coordinates of the starting point of the elliptical arc.
RMAJ,RMIN	Lengths of the semi-major and the semi-minor axis respectively.
ANGLE	Angle in degrees between the major axis and the x-axis.
THO,THF	Relative to the ANGLE, radial angles of starting and ending points of the arc.
KODE	Pen control code:

KODE	Meaning
3	Pen UP from the current position to (XBEGIN,YBEGIN).
2	Pen DOWN from the current position to (XBEGIN,YBEGIN).

Subroutine: *CALL RECT(XBEGIN, YBEGIN, DEPTH, WIDTH, ANGLE, KODE)*

Function: To plot a rectangle.

Parameters:

XBEGIN,YBEGIN	Coordinates of the lower left corner of the rectangle before rotation.
DEPTH,WIDTH	The y-x measurements of recitangle size, before rotation.
ANGLE	Angle of rotation in degrees about the point (XBEGIN,YBEGIN).
KODE	Pen control code:

KODE	Meaning
3	Pen UP before moving to the starting point.
2	Pen DOWN before moving to the starting point.

Subroutine: *CALL POLY(XBEGIN, YBEGIN, SIDE, NSIDE, ANGLE)*

Function: To draw an equilateral NSIDE-side polygon.

Parameters:

XBEGIN,YBEGIN	Coordinate of the starting vertex the polygon.
SIDE	Length of each side of the equilateral polygon.
NSIDE	Number of sides.
ANGLE	Angle in degrees of the first side with respect to the x-axis.

Table 6.8 CalComp Subroutines - Simple Geometric Patterns

Example 3: Example of AXIS calls and their outputs are shown in Figure 6.9:

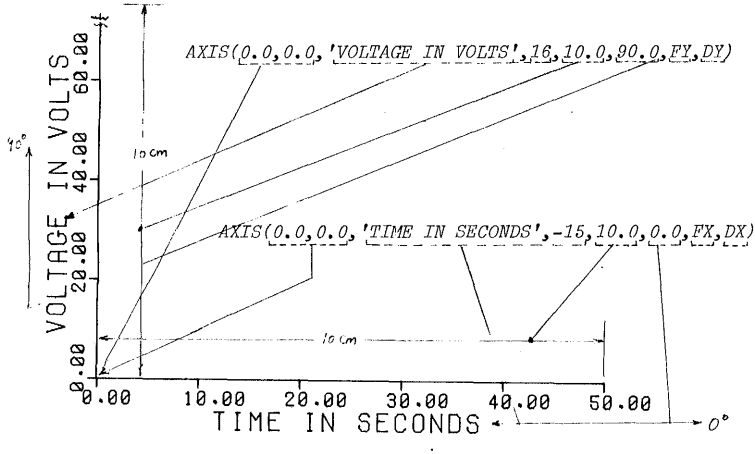


Figure 6.9 Result of the AXIS Subroutine

Example 4: A set of x-y data has been saved as DA.DAT from a CSMP run. The data format is (2E), one x-y coordinates per record. There are 46 data points. Plot a curve and label the x-axis with "TIME IN MILLISECONDS", and y-axis with "VOLTAGE IN VOLTS".

Program:

```
      REAL X(48),Y(48)                        ! DIMENSION 2 more than array size
      OPEN(UNIT=1,FILE='DA.DAT')
      READ(1,100)((X(I),Y(I)),I=1,46)         ! Read in data
100   FORMAT(2E)
*
      CALL GRAPH(15.0,15.0)                   ! Initialize plotter
      CALL ORIGIN(2.0,2.0)                    ! Move origin
      CALL SCALE(X,10.0,46,1)                 ! Scale x-axis
      CALL SCALE(Y,10.0,46,1)                 ! Scale y-axis
      FIRSTX=X(47); DELTAX=X(48)              ! Scale factor for x-axis
      FIRSTY=Y(47); DELTAY=Y(48)              ! Scale factor for y-axis
      CALL AXIS(0.0,0.0,'TIME IN MILLISECONDS',   ! Plot x-axis, label
     1  -20,10.0,0.,FIRSTX,DELTAX)            !  below x-axis
      CALL AXIS(0.0,0.0,'VOLTAGE IN VOLTS',   ! Plot y-axis, label to
     1  16,10.0,90.0,FIRSTY,DELTAY)           !  the left of y-axis
      CALL LINE(X,Y,46,1,0,0)                 ! Plot the curve
      CALL SYMBOL(4.0,10.0,0.5,               ! Plot the title
     1  'FILTER OUTPUT',0.,13)
      CALL ENDPAG                             ! Job done; terminate
      END
```

After applying the execution command and the plot command afterwards, the output was obtained from the plotter as shown in Figure 6.10.

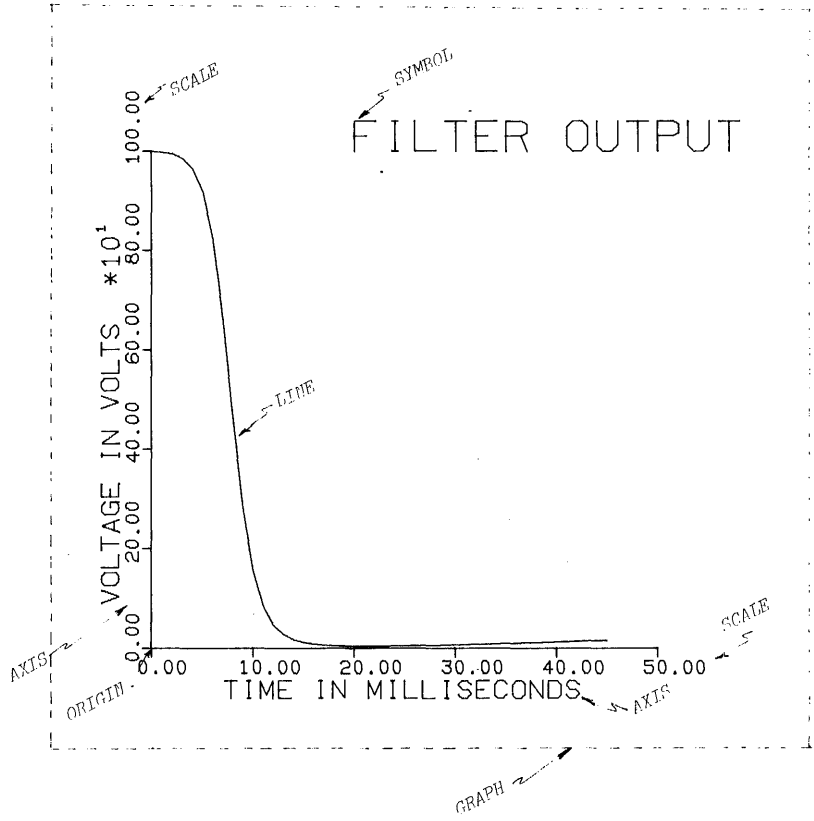


Figure 6.10 CalComp Plotter Output of Example 4, Page 257
Subroutine Names Show Their Respective Results.

A PRIMER OF GRAPHICS SOFTWARE FOR GRAPHICS TERMINALS

Since there has not yet been any standardization of graphics software, selection of a software for the PRIMER must depend on how widely its compatible graphics hardwares are available. While there are many manufacturers in the graphics area, perhaps the most widely used products are the Tektronix graphic terminals such as Models 4006, 4010, 4012/4013 or 4014/15. Serving these terminals is a collection of Tektronix-supplied software called the PLOT-10 system. It includes a basic set of graphics terminal subroutines called the Terminal Control System (TCS), an Advanced Graphing II System (AG II), the Interactive Graphing Package, The "Easy Graphing" Package, the Interactive Graphic Library (IGL), and utility routines. The TCS contains a group of FORTRAN-callable subroutines that is the basic building blocks for graphic operations and is supported at the University of Pittsburgh. The PRIMER part will only cover the TCS system. When a FORTRAN program, assuming named as PRGM.FOR, containing the TCS calls is executed, the execution command is:

```
.EXECUTE  PRGM, PRG:TEKLIB/LIB
```

where PRG:TEKLIB is the PLOT10 library (incuding AG II package) stored. This command must be given on a Tektronix graphics terminal (such as model 4010), or on a PLOT-10 compatible terminal.

6.8 Basic principle of a Graphics Terminal

A typical graphics terminal is the Tektronix model 4010-1 (with various suffix designations). Standard interface makes the terminal compatible to the computer like a conventional terminal. Thus the communication between the computer and 4010 is in the ASCII codes. The graphics terminal can also be operated as a graphics display device when a special control signal is given. Thus a 4010 can operate in (1) "alpha mode" or (2) graphics mode.

In the alpha mode, the 4010 operates as a conventional CRT terminal.

In the graphic mode, the 4010 directs an electronic beam to any of the 1024 addressable points in each axis.

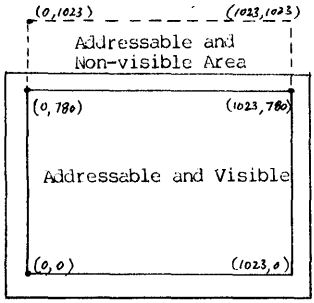


Figure 6.10
Screen Size and Coordinates

Thus the screen area has a coordinate system of 1024x1024 as shown in Figure 6.10. In the Y-axis, only a range of ordinate of 0-780 is within viewing area. An ordinate of 781-1023 range is addressable, but it cannot be displayed. Thus each of the abscissa and ordinate information will require 10 binary digits. Since the codes for communication between the computer and the terminals are ASCII codes, these binary coordinates information needed for display must be "camouflaged" as ASCII coded characters. As each ASCII character is coded in 7 bits, 4010 uses two ASCII characters (14 bits, with 4 bits to spare) to code the x-abscissa, and another two for the ordinate.

Thus, for every coordinate information, 4 ASCII characters are transmitted from the computer to the graphics terminal.

When the terminal is in the graphics mode, the hardware will take a four-character group, strip away the most significant 2 bits from each character, and combine the remaining bits into the ordinate and the abscissa data. Figure 6.11 shows how a four-character group "SPACE", "[", "=", and "DELETE" is decoded into a coordinate of (959,27).

Character	ASCII Code	5-bit Byte	Decoded Coordinates
SPACE	0100000	00000	
[	1011011	11011	Y = 0000011011 = Decimal 27
=	0111101	11101	
DELETE	1111111	11111	X = 1110111111 = Decimal 959

Figure 6.11 Decoding of ASCII codes into Screen Coordinates

The x-y information is then fed to the deflecting circuits of the CRT to move the beam. It is apparent that the coding and decoding of graphic information is very tedious. Fortunately, the coding of graphic information for transmission to the terminal is done by the PLOT10 software, and the decoding for display is done by the graphics terminal hardware. Also, unfortunately, these software and hardware in the graphics industry are not yet standardized.

Example: Set the graphics terminal on LOCAL.
(1) Press PAGE key to erase the screen.
(2) Enter into graphic mode by pressing CTRL-SHIFT-M key.
(3 keys pressed down together)
(3) Enter the following 4-character groups:
(each key pressed in sequence)

SPACE	DELETE SPACE	@	(Y=31 and X=0)
7	DELETE SPACE	@	(Y=767 and X=0)
7	DELETE ?	_	(Y=767 and X=1023)
SPACE	DELETE ?	_	(Y=31 and X=1023)
SPACE	DELETE SPACE	@	(Y=31 and X=0)

These steps should trace a diagonal cross on the screen.

To switch from the alpha-mode to graphics mode, the computer will send an ASCII character CONTROL-SHIFT-M; to switch back to the alpha-mode, an ASCII code of ESC-FORMFEED.

6.9 Terminology

In the presentation of the PRIMER that follows, some terminology will be explained here first.

(1) A/N This is an abbreviation for "alphanumeric."

(2) A/N cursor A blinking marker to show the next character print position. This is the same as in any CRT terminal.

(3) Graphic cursor The graphic cursor on 4010 is a cross-hair cursor that may be controlled by two thumb wheels. This is used as positional input during graphic mode.

(4) Home position The upper-left corner screen location at which the first character of a page is normally printed. Same as a conventional CRT terminal.

(5) Origin The coordinate represented by (0,0). The screen origin is located at the lower-left corner. Virtual space has its origin at its center.

(6) Raster unit The distance between two adjacent points on the screen. This is the basic resolution element of the terminal.

(7) Screen coordinates The set of points which constitutes the screen. Range of the screen coordiantes is from 0 to 1023 for both x and y. The visible range for y is from 0 to 780.

(8) Storage tube A CRT which will maintain a display, once written, for an indefinite priod unitl it is erased. The Tektronix 4010 is of such type. The stored display may be appended by additional display on the same picture. It cannot accommodate a subtraction of displayed information. Therefore, to even make a very minor non-appending modification of a display, the current display must be erased and redrawn.

(9) Vector When the beam is moved from one point to another, the changes in the coordinates are translated into the voltage changes that applied to the deflection plates of the CRT. The changes are made into a linear function of time, and therefore the movement of the beam on the screen will be a straight line. This is called a hardware vector generator. The vector is then a line segment. A vector may be generated with the beam either ON or OFF, which is analogous to a plotter pen position DOWN or UP.

(10) Absolute coordinates and relative coordiates The absolute coordinates use either the screen origin or the virtual space origin as reference. The relative coordinates are incremental values in x and y (positive or negative), using the current point as a reference.

(11) ADE (ASCII Decimal Equivalent) Code These are same as the conventional ASCII codes, except that they are represented in decimal integers. For example, upper case "A" is coded as decimal 65. See Table 6.4 (Chapter 6).

6.10 Screen Graphics and Virtual Graphics

First, let us establish an analogous situation. When we use a graph paper to plot a curve, the graph paper size is fixed. Yet it will be capable to plot values of any range merely by defining two things: (1) the ranges of x and y, and (2) the scale factor for x and y. This is equivalent to "zoom" the real-world scale onto the graph paper size. Thus the measurement of data directly on the graph paper would be referred to as "direct graphics", and the zooming process is the virtual graphics. In the graphics terminal, we use the uterm "screen graphics" for the term direct graphics. The process of the virtual graphics involves a normalization and scaling of all data points and translating them into actual screen coordinates for the actual hardware display. This process is not difficult, but it is exceedingly laborious because it must be applied to all axis, all labels, all data points location, and all drawings. These laborious chores are spared by the virtual graphics software. Throught the use of the virtual graphics, it is possible to increase or decrease the

apparent picture size without having to change the data values. This effectively provides an elementary "zooming" capability mentioned above. Now we are ready to define a few more terms:

(1) Screen space and screen coordinates These are as defined in section 6.9.

(2) Screen window A rectangular section of the screen space. It is usually the section into which the virtual window is scaled and translated.

(3) Virtual space this is a user-defined, data-structured rectangular display area which is independent of the terminal.

(4) Virtual window The translation of the screen window translated into the virtual space.

(5) Virtual coordinates This is a set of point coordinates that constitues virtual space. Figure 6.12 shows the basic viewporting principle of zooming from the screen to virtual graphics.

Note that in the virtual graphics, the sizes of the screen space, virtual space, screen window, and virutal window may be individually specified. Furthermore, in each rectangular specification, the width and the height can also be individually specified. This capability leads a wide flexibility of graphics display, some of which are shown in Figure 6.13. Note that an increase in the window specification decreases the apparent picture size (zoom-out), and vice versa. Also note that when the window width and height are set not in proportion between the screen window and the virtual window, there will be a distortion of display. This distortion may not be objectional because it simply implies a scale change respectively in the x- and y-d irections. If we are drawing scaled models, then this will result in a objectionable distortion.

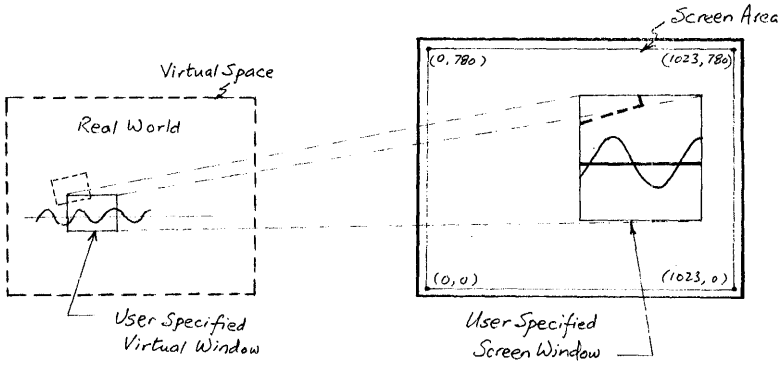
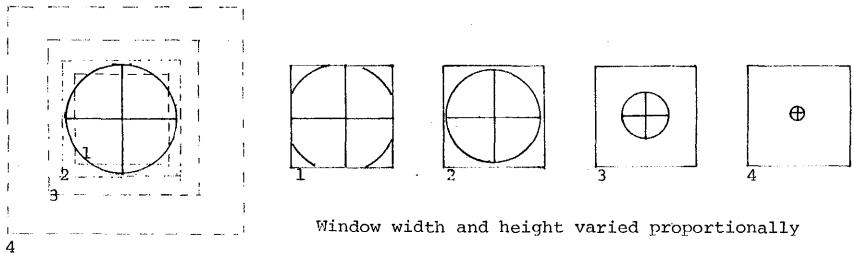
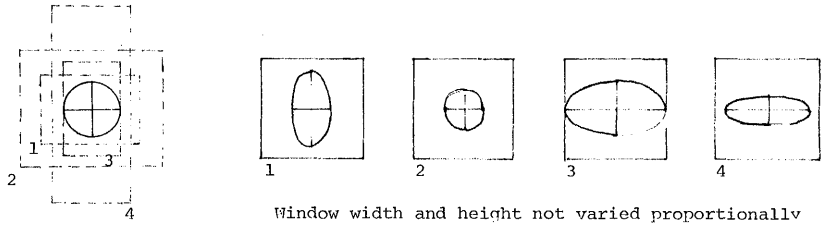
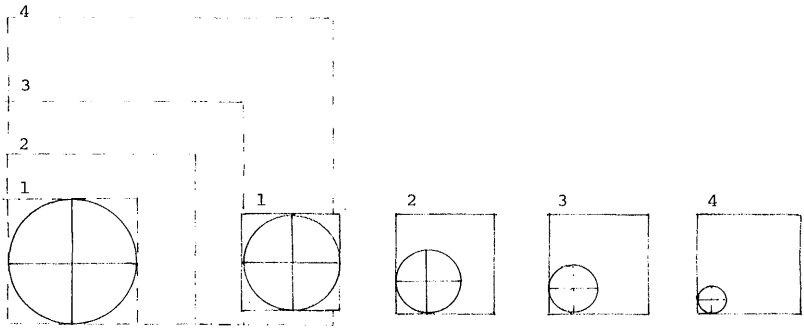


Figure 6.12 The Basic Viewporting Principle





(a) Fixed Virtual Space (Size not shown). Variable Virtual Window



(a) Fixed Virtual Window (Window 1), Variable Virtual Space

Figure 6.13 Projections by Virtual Graphics

6.11 A Basic Set of TCS Subroutines

The PLOT10 software has a hierarchial structure. At the most primitive and lowest level is a set of basic TCS soubroutines. These subroutines define the fundamental operations of the graphics. Building upon these primitive subroutines are other more advanced TCS routines. Then the advanced graphing package and other graphics software are higher-level subroutines that use the TCS routines as building blocks.

The basic TCS routines are presented in the following categories and they are respectively tabulated in five tables: (1) function control (Table 6.9), (2) screen (direct) graphics (Table 6.10), (3) virtual graphics (Table 6.11), (4) utility routines (Table 6.12), and (5) cursor operations (Table 6.13).

Subroutine:	*CALL INITT(IBAUD)*
Function:	To initialize the Tektronix terminal for a graphics session. It will turn the terminal to the graphics mode, clear the screen, set all graphic parameters to default values, set the transmission filler characters based on the baud rates, and move the beam to the screen origin.
Parameters:	
IBAUD	the transmission rate in characters/second. IBAUD=10,15,30, 120 for 110, 150, 300 and 1200 bauds respecitvely.
Subroutine:	*CALL FINITT(IX,IY)*
Function:	To terminate the graphics session and reset it to alphanumeric mode. Move the beam to a screen coordinate of (IX,IY). Typically (IX,IY) is the coordinate of the A/N home position (0,767).
Parameters:	
IX,IY	final screen coordinate of the beam position to be set
Subroutine:	*CALL ERASE*
Subroutine:	*CALL BELL*
Subroutine:	*CALL HDCOPY*
Function:	ERASE --- To erase the screen BELL --- To sound an audible alarm to alert the user. Usually used as a non-displaying prompt signal. HDCOPY --- To produce a hardcopy of the screen display
Parameters:	None

Table 6.9 PLOT10 TCS Subprograms - Function Control

Subroutine: *CALL DRWABS(IX, IY)*

Subroutine: *CALL MOVABS(IX, IY)*

Subroutine: *CALL PNTABS(IX, IY)*

Subroutine: *CALL DSHABS(IX, IY, L)*

Function: Screen graphics with absolute screen coordinates:

DRWABS --- To draw a vector from the current position to (IX,IY)
MOVABS --- To move the beam from the present position to (IX,IY)
PNTABS --- to plot a point at the position (IX,IY)
DSHABS --- To draw a dash line from the present position to (IX,IY)

Parameters:

IX,IY — the screen coordinates of the destination point
L — dash line codes:

1	5 raster units, visible
2	5 raster units, invisible
3	10 raster units, visible
4	10 raster units, invisible
5	25 raster units, visible
6	25 raster units, invisible
7	50 raster units, visible
8	50 raster units, invisible

L is a dash line code that is a concatenation of the above code numbers. For example, when L=3454, the dash line will be drawn in the pattern of 345434543454... where each code number is defined as above.

When L is given as a single digit, it is interpreted as follows:

-1	causes a move
0	causes a draw
1	alternate visible and invisible segments between data points.

Subroutine: *CALL DRWREL(IX, IY)*

Subroutine: *CALL MOVREL(IX, IY)*

Subroutine: *CALL PNTREL(IX, IY)*

Subroutine: *CALL DSHREL(IX, IY, L)*

Function: Screen graphics with relative screen coordinates:

DRWREL --- To draw a line from current position to another point with a known displacement from the current position.
MOVREL --- To move the beam from current position to another point with known displacements
PNTREL --- To plot a point at a screen displacement of (IX,IY) from the current point
DSHREL --- to draw a dash line from the current position to another point with a known x- and y-displacements

Parameters:

IX,IY — the x- and the y-displacements (in raster units) from the current position
L — the dash line code as defined above

Table 6.10 PLOT10 TCS Subprograms - Screen Graphics

Example: Screen graphics is simple to implement. It consists of the following steps:

(1) Initialize.
(2) Move the beam ("Pen UP" fashion) to the first point.
(3) Start drawing. FORTRAN statements for calculating the data points may be intermixed with the graphics statements in their natural order.
(4) Terminate the graphics session.

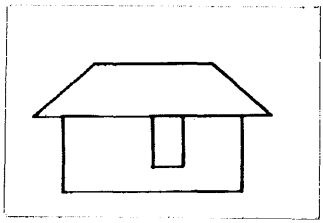


Figure 6.14 Hardcopy of the Output

The following program, when executed, will produce a drawing as shown in Figure 6.14.

```
*  Example for screen graphics
      DIMENSION IX(13),IY(13)
      DATA IX/500,900,700,300,100,800,
     1  800,200,200,600,600,500,500/
      DATA IY/400,400,600,600,400,400,
     1  100,100,400,400,200,200,400/
      CALL INITT(30)
      CALL MOVABS(IX(1),IY(1))
      DO 10 I=2,13
10    CALL DRWABS(IX(I),IY(I))
      CALL FINITT(0,767)
      END
```

Be careful not to let screen coordinates overflow. If either of the coordinates is specified with an integer larger than 1023, the true coordinate plotted on some graphic terminals will be the residue number of modulo 1024. Thus it produces a wrap-around effect. On some graphic terminals, the segment beyond the screen area will not be displayed. Furthermore, when the beam returns to the display area, it will return at the place where it goes out of bound, thus traces an incorrect line. If the line goes out of bound but still within the addressable area (i.e. Y>780 and Y<1024), the picture will be clipped for those parts outside the visible display area. These common errors are illustrated in Figure 6.15.

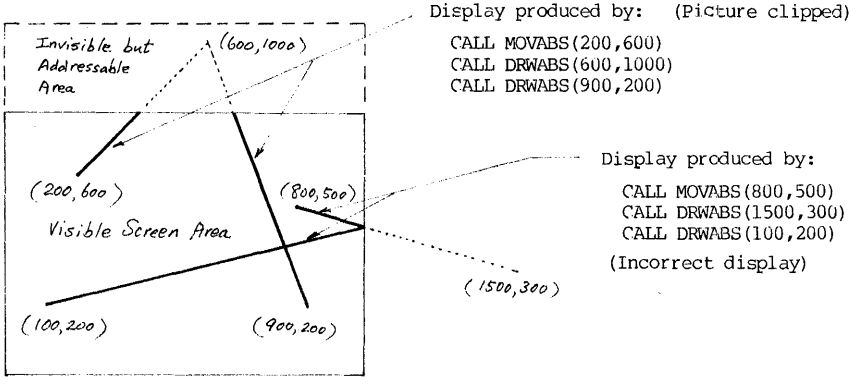


Figure 6.15 Screen Coordinate Overflow Errors

Subroutine: *CALL VWINDO(XMIN, XRANGE, YMIN, YRANGE)*

Subroutine: *CALL DWINDO(XMIN, XMAX, YMIN, YMAX)*

Subroutine: *CALL SWINDO(MINX, LENX, MINY, LENY)*

Subroutine: *CALL TWINDO(MINX, MAXX, MINY, MAXY)*

Function: To define the windows:

VWINDO and DWINDO --- To define a virtual window
SWINDO and TWINDO --- To define a screen window

Parameters:

XMIN,XMAX	the minimum and the maximum virtual x-coordinates
YMIN,YMAX	the minimum and the maximum virtual y-coordinates
XRAGNE,YRANGE	the horizontal and the vertical extents of the window

MINX,MAXX,MINY,MAXY,LENX,LENY
Similar definitions, except they are integers and they define ascreen window.

Subroutine: *CALL DRAWA(X, Y)*

Subroutine: *CALL MOVEA(X, Y)*

Subroutine: *CALL POINTA(X, Y)*

Subroutine: *CALL DASHA(X,Y,L)*

Functions: Virtual graphics in absolute virtual coordinate:

DRAWA --- To draw a line from current position to virtual coordinate (X,Y)
MOVEA --- To move the beam from current position to virtual coordinate (X,Y)
POINTA --- To plot a point at the virtual coordinate (X,Y)
DASHA --- To draw a dash line from the current position to virtual coordinate (X,Y). The dash line code is given in Table 6.10.

Parameters:

X,Y	the absolute virtual coordinates of the destination point
L	the dash line code as defined in Table 6.10

Subroutine: *CALL DRAWR(X, Y)*

Subroutine: *CALL MOVER(X, Y)*

Subroutine: *CALL POINTR(X,Y)*

Subroutine: *CALL DASHR(X,Y,L)*

Functions: Virtual graphics in relative virtual coordinate:

DRAWR --- To draw a line from current position with a virtual displacement
MOVER --- To move the beam from current position with a virtual displacement
POINTR --- To plot a point at a virtual displacement from the current point
DASHR --- To draw a line form current position with a virtual displacement

Parameters:

X,Y	the horizontal and vertical displacement respectively of the destination point from the current point.
L	the dash line code as defined in Table 6.10

Table 6.11 PLOT10 TCS Subprograms - Virtual Graphics

Example: Graphics in the virtual space may be implemented in a similar way:

(1) Initialize the graphics.
(2) specify the screen window (omitted if the entire screen is the window) and the virtual window.
(3) Move the beam to the first point using virtual coordinate.
(4) Start drawing. Statements for calculating the data points may be intermixed with the graphics statements in their natural order. Screen graphics and virtual graphics can also be intermixed.

The following program will project the drawing of Figure 6.14 and reproduce it within several virtual windows as shown in Figure 6.16.

```
*  Example for virtual graphics
      DIMENSION IX(13),IY(13),X(13),Y(13)
      DATA IX/500,900,800,300,100,800,800,200,200,600,600,500,500/
      DATA IY/400,400,600,600,400,400,100,100,400,400,200,200,400/
      CALL INITT(30)
      CALL DWINDO(50.,950.,50.,650.)
      DO 10 I=1,13
      X(I)=FLOAT(IX(I)); Y(I)=FLOAT(IY(I))
10      CONTINUE
      DO 50 K=1,3
         IF(K.EQ.1)CALL TWINDO(600,800,300,500)
         IF(K.EQ.2)CALL TWINDO(100,200,100,600)
         IF(K.EQ.3)CALL TWINDO(100,800,100,200)
         CALL MOVER(X(1),Y(1))
         DO 20 I=2,13
20          CALL DRAWA(X(I),Y(I))
50      CONTINUE
      CALL FINITT(0,767)
      END
```

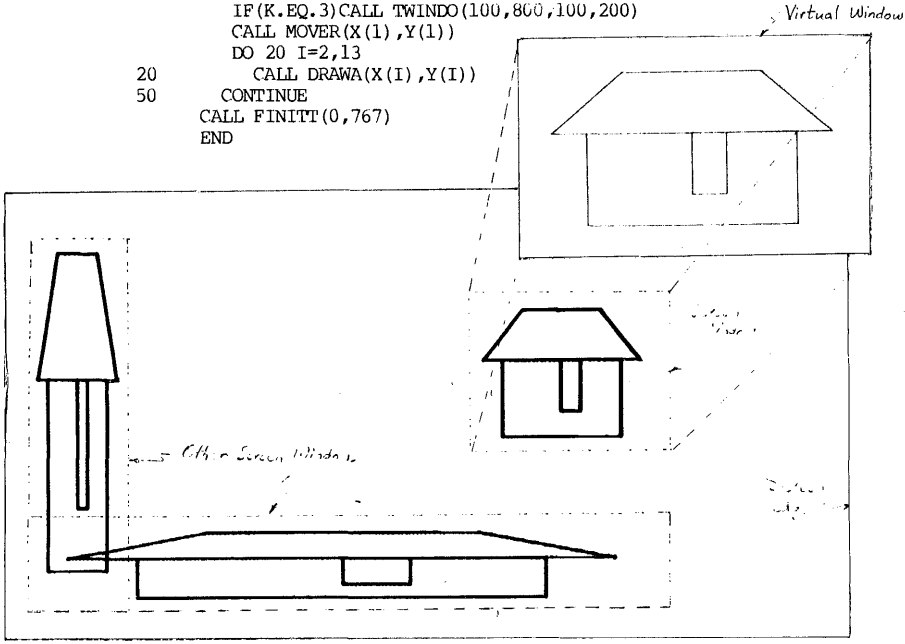


Figure 6.16 Virtual Graphics Example

Subroutine:	*CALL ANMODE*
Function:	To set the terminal to A/N output rather than having to use FORTRAN READ and WRITE statements, thus remaining in the graphics mode.
Parameters:	None

Subroutine:	*CALL ANCHO(ICHAR)*
Subroutine:	*CALL ANSTR(NCHAR,NADE)*
Function:	To produce an ASCII output on the screen. The beam position is updated after the writing. ANCHO --- To output one single character. ANSTR --- To output a string of characters
Parameters:	
ICHAR	an integer representing a 7-bit ASCII character; not a control-character.
NCHAR	number of characters in the string
NADE	an array containing the ASCII decimal integer equivalent for the characters in the string

Subroutine:	*CALL NEWLIN*
Subroutine:	*CALL CARTN*
Subroutine:	*CALL LINEF*
Subroutine:	*CALL BAKSP*
Subroutine:	*CALL HOME*
Subroutine:	*CALL NEWPAG*
Function:	To provide utility functions of the terminal NEWLIN --- To generate a line feed and carriage return LINEF --- To generate a line feed alone, no carriage return CARTN --- To generate a carriage return, no line feed HOME --- To move the A/N cursor to the home position BAKSP --- To generate a backspace NEWPAG --- To erase the screen and return the A/N cursor to the home position
Parameters:	None

Table 6.12 PLOT10 TCS Subprograms - Utility Routines

Subroutine:	*CALL SCURSR(ICHAR,IX,IY)*
Subroutine:	*CALL DCURSR(ICHAR,IX,IY)*
Function:	To retrieve the screen coordinates of the graphic cursor. Calling the SCURSR or DCURSR will activate the graphic cursor, a cross-hair line. Two thumb wheels may be used to move the line in order to position a point on the screen. The cursor position is transmitted to the computer when a keyboard character is struck. The subroutine returns with IX,IY indicating the screen coordinates of the cursor. SCURSR and DCURSR are identical routines.
Parameters:	
ICHAR	a keyboard character; an decimal integer equivalent of its ASCII code
IX,IY	the screen coordinates of the graphic cursor returned by the subroutine
Subroutine:	*CALL VCURSR(ICHAR,X,Y)*
Function:	To retrieve the virtual coordinates of the graphic cursor
Parameters:	
ICHAR	a keyboard character, represented by its ASCII code in the decimal equivalent
X,Y	the virtual coordinates of the graphic cursor

Table 6.13 PLOT10 TCS Subprograms - Cursor Operations

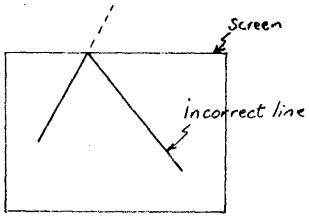


Figure 6.17 Overflow Errors in Virutal Graphics

Overflow errors in virtual graphics will appear as clipped picture at the edges of the virtual window. A more serious and difficult to detect error is committed when the beam is moved outside the virtual window by virtual coordinates and then moved back inside the window by screen coordinates. The re-entry point will be where the beam left the virtual window. Because there will not be a clipped figure, the error is difficult to detect but the re-entry line will be incorrectly drawn. This is illustrated in Figure 6.17 on the left.

6.12 Interactive Graphics

Time-sharing mode of computer operation puts a user in direct contact with the computer. Now graphics opens a new world with its clarity of information. Interactive graphics is then a natural result of combining the best of these two operational modes.

There are some problems in constructing the computer-user dialogues in the graphic mode, however. In a conventional terminal, the dialogues are constructed by READ/WRITE statements and the decision (IF) statements following the dialogue to determine the next step. All these dialogues are displayed on the user's terminal.

In a graphics terminal, such dialogues are still applicable when the terminal is in an alphanumerical mode. Once it gets into the graphics mode, the drawing will be in progress, and dialogues should not appear on the screen because they will spoil the picture. One possible remedy is to move the beam outside the visible region; return to the A/N mode; tell the computer your part of the conversation (by a FORTRAN READ statement); then return to the graphics mode. Another is to use the ASCII input subroutines, such as TINSTR or TINPUT, which will be explained in a later section. The most effective way is by using the "menu" graphics for the interaction.

Let us use the 4010 screen for illustration. Suppose we arbitrarily assign the vertical area at the right edge of the screen as the "menu" area. Within the menu area, subsections are assigned to pre-designed options and/or decisions. By calling the cursor and moving it to within the desired subsection, the user's intention is transmitted to the computer without spoiling the picture. In addition, the bell can be used as an audible prompt signal to alert the user. "Menu graphics" basically makes the use of these ingredients. Now we will illustrate by means of an example on the Tektronix 4010:

Example: Design an interactive graphics program that will make a logic circuit diagram consisting of all NAND gates. In other words, the screen will be used as a drawing board, the cursor as a pen.

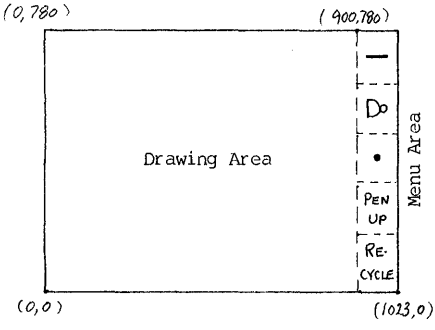


Figure 6.18 Menu Design

First, set aside the area on the right edge of the screen as the menu area. The menu area is a rectangle with the upper left corner at the screen coordinate (900,780) and the lower right corner at (1023,0). Then the menu area is equally divided horizontally into five subsections. This is shown in Figure 6.18. Five options are designed into the program:

(1) Option 1 Draw a solid line.
(2) Option 2 Draw a NAND symbol.
(3) Option 3 Draw a small dot, as interconnecting point of two lines.
(4) Option 4 Move the beam to a new position.
(5) Option 5 Make a hardcopy of the screen display and recycle the program.

To simplify the drawing, we will set up a default option. If the graphic cursor is positioned outside the menu area, the option is the same as the one chosen previously, and the cursor position specifies the end beam position. Thus only when the option is to be different from the last choice for the cursor to return to the menu area.

The design specification of the interactive program can then be summarized into the following steps of operations:

(1) The program is initialized by blanking the screen, and draw the menu. The cursor appears on the screen, and the beam is in PEN-UP mode (or, OPTION=4). Move the cursor, using two thumb wheels on the keyboard, to a point in the drawing area and mark this position by pressing the carriage return (CR) key. The beam is now positioned at the starting point of the diagram.

(2) The bell beeps to alert the user to make a choice of next option.

   a. Cursor will reappear.
   b. Move and position the cursor to the menu area, and press CR key to select an option.
   c. Move the cursor to the drawing area, and press the CR key to execute the option.

   If the option chosen is the same as the last one, the steps "a" and "b" may be omitted. Repeat step 2 until the drawing is finished.

(3) When the drawing is finished, make a hardcopy by pressing the HDCOPY switch on the terminal. Choose option 5 (RECYCLE) to make the next drawing, or press CTRL-C to exit.

The program listing is shown below:

```
* ************************************
*                                    *
*       FILENAME:    DRAW.FOR        *
*                                    *
* ************************************
*
*  An interactive menu graphics program to draw an
*    All-NAND logic circuit diagram
*
*  Implemented for Tektronix 4010 series terminals
*
      CALL INITT(30)
10    CALL START
      JOB0=4
20    CALL OPTION(JOB1,IX,IY)
      IF(JOB1.EQ.0)GOTO 50
      IF(JOB1.EQ.5)GOTO 10
      CALL BELL
      CALL SCURSR(13,IX,IY)
50    IF(JOB1.GT.1)CALL MOVABS(IX,IY)
      IF(JOB1.EQ.1)CALL DRWABS(IX,IY)
      IF(JOB1.EQ.2)CALL NAND(IX,IY)
      IF(JOB1.EQ.3)CALL DOT(IX,IY)
      JOB1=JOB0
      GOTO 20
      END
```

```
************************
*  SUBROUTINE   START  *
************************
*
*  TO INITIATE THE MENU GRAPHICS
*
      SUBROUTINE START
      DIMENSION K1(6),K2(7)
      DATA K1/'P','E','N',' ','U','P'/
      DATA K2/'R','E','C','Y','C','L','E'/
      CALL NEWPAG
      CALL MOVABS(900,780)
      CALL DRWABS(1023,780)
      CALL DRWABS(1023,0)
      CALL DRWABS(900,0)
      CALL DRWABS(900,780)
      IY1=780
      DO 10 I=1,4
      IY1=IY1-156
      CALL MOVABS(900,IY1)
      CALL DSHABS(1023,IY1,12)
10    CONTINUE
      CALL MOVABS(925,702)
      CALL DRWABS(1000,702)
      CALL NAND(950,546)
      CALL DOT(960,390)
      CALL MOVABS(910,220)
      CALL A1OUT(6,K1)
      CALL MOVABS(910,65)
      CALL A1OUT(7,K2)
      RETURN
      END
************************
*  SUBROUTINE  OPTION  *
************************
*
*  Menu selection of options
*      Returned values:
*        JOB1 = 0    Same job as before
*        JOB1 = 1    Draw line
*        JOB1 = 2    Draw an NAND symbol
*        JOB1 = 3    Draw a dot
*        JOB1 = 4    Move beam, no drawing
*
*   IX,IY information needed for JOB1=0
*
      SUBROUTINE OPTION(JOB1,IX,IY)
      CALL BELL
      CALL DCURSR(13,IX,IY)
      IF(IX.LT.900)JOB1=0
      IF(IX.LT.900)RETURN
      IF(IY.GT.624)JOB1=1
      IF((IY.LE.614).AND.(IY.GT.468))JOB1=2
      IF((IY.LE.468).AND.(IY.GT.312))JOB1=3
      IF((IY.LE.312).AND.(IY.GT.156))JOB1=4
      IF(IY.LE.156)JOB1=5
      RETURN
      END
************************
*  SUBROUTINE    NAND  *
************************
*
*  To draw an NAND symbol at (IX,IY)
*
      SUBROUTINE NAND(IX,IY)
      INTEGER INX(16),INY(16)
      DATA INX/0,13,9,3,1,3,3,1,-1,
     1  -3,-3,-1,-3,-9,-13,0/
      DATA INY/-25,3,9,13,3,1,-1,-3,
     1  -3,-1,1,3,13,9,3,-25/
      CALL MOVABS(IX,IY)
      DO 10 I=1,16
10    CALL DRWREL(INX(I),INY(I))
      CALL MOVABS(IX,IY)
      RETURN
      END
************************
*  SUBROUTINE     DOT  *
************************
*
*  To draw a intersecting dot
*
      SUBROUTINE DOT(IX,IY)
      INTEGER INX(8),INY(8)
      DATA INX/-1,-2,-2,-1,1,2,2,1/
      DATA INY/2,1,-1,-2,-2,-1,1,2/
      CALL MOVABS(IX,IY)
      CALL PNTABS(IX,IY)
      DO 10 K=1,3
      IK=K
      CALL MOVABS(IK+IX,IY)
      CALL DRWREL(-IK,IK)
      CALL DRWREL(-IK,-IK)
      CALL DRWREL(IK,-IK)
      CALL DRWREL(IK,IK)
10    CONTINUE
      DO 20 I=1,8
20    CALL DRWREL(INX(I),INY(I))
      CALL MOVABS(IX,IY)
      RETURN
      END
```

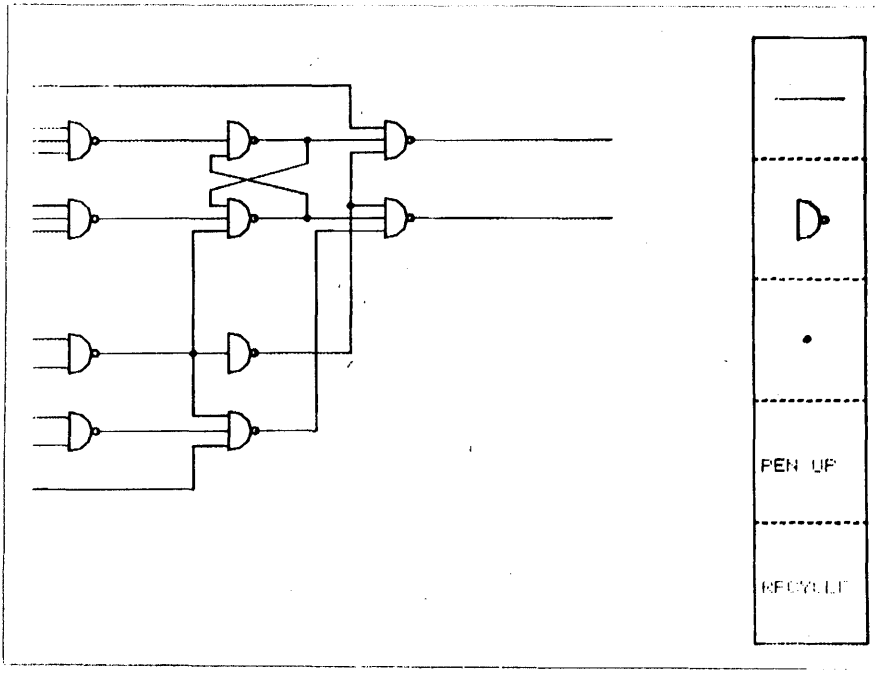


Figure 6.19 Output from the Interactive Program

Figure 6.19 shows a sample of the result from this interactive program.

This example shows a very simple case of menu graphics. In a more advanced application, both the software and the hardware support will become more sophiscated. The thumb-wheel controlled cursor will be replaced by a light pen, a joystick or a trackball. The menu will not occupy the same screen as the display. Some terminal has several display xy-plane (in the z-axis), and it becomes possible to put the display on one plane, and the menu and the cursor on another. The display planes are superimposed on the screen surface. Some graphic system has separate digital grphic tablet that has the same coordinate ranges as those of the screen, and the tablet may be used as a graphic positional input device. The menu in such a case may be designed as an overlay that has a drawing of the menu on it and is placed on top of the tablet. A light pen or a mechanical cursor device may be used to make a selection of options.

6.13 A Summary of Other TCS Subprograms

In addition to the basic set of TCS subprograms presented in the last section, there are other subprograms tht are useful. They are again divided into several categories for summarizing tabulations: (1) rescaling graphic output (Table 6.14), (2) virtual graphics and beam status (Table 6.15), (3) output utilities (Table 6.16), (4) coordinate transformation (Table 6.17), and (5) ASCII input/output (Table 6.18).

These tables now follow.

Subroutine:	*CALL RSCALE(FACTOR)*
Function:	To rescale a virtual display by a virtual factor
Parameters:	
FACTOR	the rescaling factor relative to the original size of the display
Subroutine:	*CALL RROTAT(DEGREE)*
Function:	To rotate a virtual display by an angle relative to its original display position.
Parameters:	
DEGREE	angle in degrees for the rotation
Subroutine:	*CALL RESET*
Subroutine:	*CALL RECOVR*
Function:	RESET --- same as INITT, but no erasure of screen RECOVR --- To update the terminal hardware to match the terminal status values
Parameters:	None

Table 6.14 PLOT10 TCS Subporgrams - Rescaling the Graphic Output

Subroutine:	*CALL SEETW(MINX, MAXX, MINY, MAXY)*
Subroutine:	*CALL SEEDW(XMIN, XMAX, YMIN, YMAX)*
Function:	SEETW --- To find the current values of the screen window SEEDW --- To find the current values of the virtual window
Parameters:	These are returned parameters:
MINX,MAXX,MINY,MAXY	the screen coordinates that define a screen window
XMIN,XMAX,YMIN,YMAX	the virtual coordinates that define a virtual window
Subroutine:	*CALL SEEREL(RCOS, RSIN, SCALE)*
Function:	To return the values of the common variables used by the relative virtual routine to scale and rotate vectors.
Parameters:	
RCOS	the cosine of the rotating angle
RSIN	the sine of the rotating angle
SCALE	the multiplier used for scaling
Subroutine:	*CALL SEETRN(XFAC, YFAC, KEY)*
Function:	To return the values of the common variables set by the window and transformation routines
Parameters:	
XFAC,YFAC	the x and y scale factors respectively
KEY	the transformation code: 1=linear; 2=log; 3=polar
Subroutine:	*CALL SEELOC(IX, IY)*
Function:	To locate on the screen the last positin of the graphic beam
Parameters:	
IX,IY	the screen coordinates of the beam

Table 6.15 PLOT10 TCS Subprograms - Virtual Graphics and Beam Status

Subprogram: *variable = LINWDT(NCHAR)*

Subprogram: *variable = LINHGT(NLINE)*

Input Prameters:

NCHAR	number of characters
NLINE	number of lines

Output Parameters:

LINWDT	width measurement in raster units
LINHGT	height measurement in raster units

Subprogram: *variable = KIN(RIN)*

Subprogram: *variable = KCM(RCM)*

Input Parameters:

RIN,RCM	input parameters in inches or centimeters respectively

Output Parameters:

KIN	number of raster units equivalent to the input RIN
KCM	number of raster units equivalent to the input RCM

Subroutine: *CALL SETMRG(MLEFT,MRIGHT)*

Function: To set the left and right margins

Parameters:

MLEFT,MRIGHT	screen coordinates for the left and right margins respectively

Table 6.16 PLOT10 TCS Subprograms - Output Utilities

Subroutine: *CALL LINTRN*

Function: To reset to linear scaling (from log or polar scaling)

Parameters: None

Subroutine: *CALL LOGTRN(KEY)*

Function: To define a semi-log or a log-log scale

Parameters:

KEY	code for the log scaling: KEY=1 semi-log, x-axis on log scale KEY=2 semi-log, y-axis on log scale KEY=3 log-log scale

---

Subroutine: *CALL POLTRN(ANGMIN, ANGMAX, RSPRS)*

Function: To set up polar virtual window

Parameters:

ANGMIN,ANGMAX	the minimum and the maximum angles relative to the horizontal for the display
RSUPRS	the radius suppression factor

Subroutine: *CALL DRAWSA(X, Y)*

Subroutine: *CALL DRAWSR(RX, RY)*

Subroutine: *CALL DASHSA(X, Y)*

Subroutine: *CALL DASHSR(RX, RY, L)*

Function: To drawline segment while the polar coordinate transformation is in effect:

DRAWSA --- To draw a segment of line, given a virtual coordinates of the end point
DRAWSR --- To draw a line segment given the virtual relative coordinates to the current beam position.
DASHSA --- Same as DRAWSA except in dash line
DASHSR --- Same as DRAWSR except in dash line

Parameters:

X,Y	virtual coordinates of the end point of the line
RX,RY	virtual coordinates relative to the present beam position
L	the dash line code (See Table 6.10)

Table 6.17 PLOT10 TCS Subprograms - Coordinate Transformations

Output of ASCII Characters - Beam not updated for CTRL-Character Output	
Subroutine:	*CALL ANCHO(ADE)*
Subroutine:	*CALL ANSTR(NCHAR, ADES)*
Function:	The ASCII characters are given in ADE codes.
Parameters:	
ADE	an integer constant, representing the ADE code of the character. The character must not be a control-character.
ADES	an integer array, with each an ADE code for a character. The array contains no control-cahracters.
NCHAR	number of characters
Subroutine:	*CALL A1OUT(NCHAR, ASC1S)*
Subroutine:	*CALL AOUTST(NCHAR, ASC5S)*
Function:	ASCII characters are given in FORTRAN "A" format. Not for CTRL-characters.
Parameters:	
NCHAR	number of characters. For A1 format, NCHAR = number of characters. For A5 format, NCHAR = 5*(number of elements in the array).
ASC1S	an ASCII array, with each element in A1 format
ASC5S	an ASCII array, with each element in A5 format, left-justified
Subroutine:	*CALL TOUTPT(ADE)*
Subroutine:	*CALL TOUTST(NCHAR, ADES)*
Function:	Characters are given in ADE codes. For CTRL-characters only, because the beam does not move after the output.
Parameters:	
NCHAR	number of characters
ADE	an integer constant, representing the control-character in ADE code
ADES	an integer array, each element a control-character in ADE code

To Store Input ASCII Characters from Keyboard	
Subroutine:	*CALL A1IN(NCHAR, ASC1S)*
Subroutine:	*CALL AINST(NCHAR, ASC5S)*
Function:	To accept ASCII inputs from the keyboard and store them as variable or array values in FORTRAN "A" format.
Parameters:	
NCHAR,ASC1S,ASC5S	defined the same way as in A1OUT and AOUTST subroutines above.
Subroutine:	*CALL TINPUT(ADE)*
Subroutine:	*CALL TINSTR(NCHAR, ADES)*
Function:	To accept ASCII inputs from the keyboard, and store them as variable or array values as integer ADE codes.
Parameters:	
NCHAR,ADE,ADES	defined in the same way as in TOUTPT and TOUTST above.

Table 6.18 PLOT10 TCS Subprograms - Input/Output of ASCII Characters

THREE DIMENSIONAL DISPLAYS

Three-dimensional graphics is one of the most challenging topics in the computer graphics research and applications. It involves a study of surfaces and solids, their geometrical formulations, their interrelationship expresssion by means of some language description, perspective, projection and hidden surface identifications.

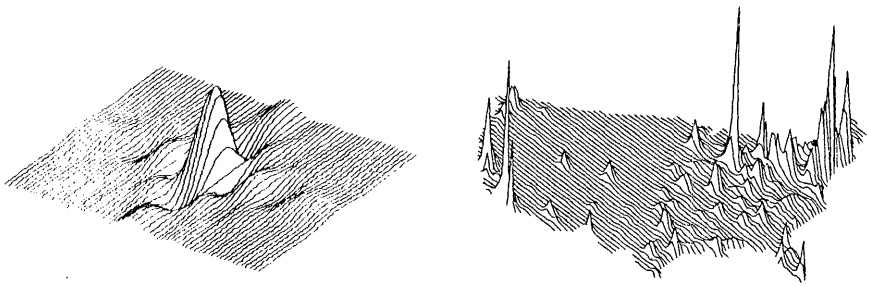
The discussion here will only concern with a very simple and small aspect of the field. We will simply be concerned with how to display a two-dimensional function in the general form of z=f(x,y) for a range of x's and a range of y's. For example, a relief map would be a display model showing the height as a function of longitudes and latitudes.

6.14 Three Dimenional Displays

In the materials covered in this chapter, the display was mainly in two dimensions. In other words, the mathematical formulation of the function is y=f(x), x=f(y), or f(x,y)=constant.

For the type of function z=f(x,y), a 3-dimensional display is required, and the z-axis is need to display "z".

There are several ways to display such functions. One is displaying it as a contour or relief map, with x,y to be spatial coordinates. Figure 6.20 shows two outputs available on DEC-10.



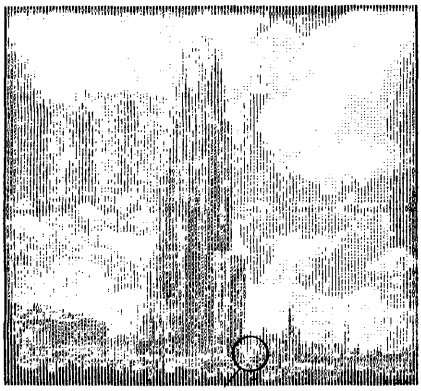
(a) 2-Dimensional Fourier Spectrum Sample Output from VERPLT (Ref.18)

(b) 1970 U.S. Census Sample Output from ASPEX (Ref.16)

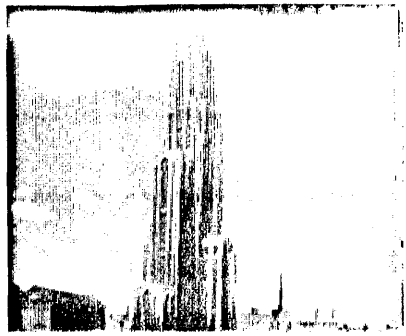
Figure 6.20 3-D Displays by Contour Plotting

The other way is to interprete z's as light intensities and plot the z-function as an image. Figure 6.21 shows some samples of image output on DEC-10. The gray scales (the gradation of shades) in the output are accomplished either by an over-print techniques (on a conventional printer) or by a dot-matrix technique (on a dot matrix printer, such as the VERSATEC printer).

(a) Over-print Technique



(b) Dot-matrix Technique



Z-function values in the circle:

158	188	182	177	176	170	163	158
161	164	158	141	108	166	174	138
151	142	129	121	111	151	156	105
142	132	119	125	100	115	131	97
114	93	106	104	81	90	132	62
95	55	83	58	61	64	104	70
66	51	71	54	54	49	81	53
45	52	57	47	48	45	56	25

Z-Value	Overprint	Dot-matrix
105	X -	⠶
25	M W X	⣿

Figure 6.21 3-D Displays by Image Plots

EXERCISES

1. Reproduce your signature on a CalComp plotter. Write a subprogram so that you may sign your plots. There should be control on where the signature should start (specify starting point coordinates), how high it should be and how wide it should be.

2. Repeat problem 1 on the Tektronix 4010.

3. Write and implement a subroutine CIRC(X,Y,R) for (a) printer graphics, (b) CalComp graphics, and (c) Tektronix 4010. The parameter X,Y are virtual coordinates for the center, and R is the radius.

4. Set up a data file for about 50 data points. Use the data file to produce plots on the printer, on the CalComp plotter, and on the Tektronix 4010. The data file may be generated by taking a function and calculating its values for a range of the independent variable.

5. Design an interactive graphic program so that you may use the cursor of Tektrnoix to draw a transistor circuit diagram. Design a menu so that you have selections of symbols of transistors, diodes, resistors, inductors, and capacitors.

6. Reproduce the University logo on the printer, on the plotter and on the Tektronix 4010. Write it up as a subroutine with parameters of size-control.
7. Practice on various type of axis constructions: linear axis and logarithmic axis, using some arbirtrary data set.

8. Produce a graphic creative design on a graphic device.

Note: CalComp plotter is a slow device, and a typical plotting job takes minutes. Therefore, most installations have restrictive regulations for student usage and require special arrangement or request from the instructor. Consult your local rules and regulations for instructional usages.

REFERENCES

1. TUTORAIL: COMPUTER GRAPHICS, Kellogg S. Booth, Editor, IEEE Catalog EHO-147-9, The Institute of Electrical and Electronics Engineers, Inc., New York, New York; 1979.

2. "Computer Displays," by Ivan E. Sutherland, Scientific American, Vol. 222, No.6 (June, 1970), pp. 36-41.

3. "A PROPOSED GRAPHICS STANDARD CORE SYSTEM", by SIGGRAPH Committee, ACM, Computing Surveys, Volume 10, No.4, December, 1978.

4. METHODOLOGY IN COMPUTER GRAPHICS, R. A. Guedj and H. A. Tucker, Editors, North-Holland Publishing Company, New York, New York; 1979.

5. Help File PRG:GRAPH.HLP, the Computer Center, University of Pittsburgh, Pittsburgh, Pennsylvania; 1980.

6. Help File PRG:GRAFIC.HLP, the Computer Center, University of Pittsburgh, Pittsburgh, Pennsylvania; 19xx.

7. WORKSHOP NOTES ON COMPUTER GRAPHICS, T. W. Sze and A. R. Modarressi, School of Engineering, University of Pittsburgh, Pittsburgh, Pennsylvania; April, 1975.

8. INTRODUCTION TO THE CALCOMP PLOTTER, DEC-10 Pitt Software-2, the Computer Center, University of Pittsburgh, Pittsburgh, Pennsylvania; October, 1976.

9. INTRUCTION MANUAL FOR DIGITAL INCREMENTAL PLOTTER, Model 936, California Computer Products, Inc., Anaheim, California; 1979.

10. CAMCOMP PLOTTER SUBROUTINES, the Computer Center, University of Pittsburgh, Pittsburgh, Pennsylvania; October, 1973.

11. CALCOMP PROGRAMMING FOR DIGITAL PLOTTER, by R. T. DeLorm and L. Kersten, University of Nebraska Press, Lincoln, Nebraska.

12. 4010 AND 4010-1 USERS MANUAL, Tektronix, Inc., Beaverton, Oregon; 1972.

13. PLOT-10 TERMINAL CONTROL SYSTEM, USER'S MANUAL, Tektronix, Inc., Beaverton, Oregon; April, 1980.

14. PLOT-10 ADVANCED GRAPHING II, USER'S MANUAL, Tektronix, Inc., Beaverton, Oregon.

15. INTERACTIVE COMPUTER GRAHICS, Wolfgang K. Giloi, Prentice-Hall Inc., Englewood Cliffs, New Jersey; 1978.

16. ASPEX USER'S REFERENCE MANUAL, Mark Hanson, Laboratory for Computer Graphics and Spatial Analysis, Harvard University, Cambridge, Massachusetts; 1978.

17. Technical Report TM-7801, IMAGE PROCESSING PROGRAMS, Mike M. Lee and John Todhunter, Pattern Recognition Laboratory, University of Pittsburgh, Pittsburgh, Pennsylvania; 1978.

18. VERSATEC 3-D PLOTTING ROUTINE, Herbert Y. H. Yang, Pattern Recognition Laboratory, University of Pittsburgh, Pittsburgh, Pennsylvania; 1979.

19. IMPROC HELP FILE, An Image Processing System for DEC-10, T. W. Sze, University of Pittsburgh, Pittsburgh, Pennsylvania; 1980.

CHAPTER 7

SELECTED SERVICE PROGRAMS AND PROCEDURES

Several service programs and procedures will be presented in this chapter. They are: PIP (Peripheral Interchange Program), SORT, RUNOFF, OPRSTK (Operation Stacker), and the Virtual Memory. These programs and procedures perform a variety of service functions.

PIP

7.1 Introduction

The PIP (Peripheral Interchange Program) program transfers data and program files from one standard input/ouput device to another. Unless explicitly specified to be deleted, a file at the source remains unchanged and undisturbed. During these operations, simple file editing and magnetic tape control may be performed. It is undoubtedly one of the most useful service programs. Since it deals with devices and files, several related terms should be reviewed first:

(1) Physical device name Each input or output peripheral device associated with the System has a standard physical device name so that it can be referred to consistently. The format of a physical device name is:

DEVnnn:

where: DEV = three-character abbreviation assigned to a class of devices, for example, LPT for all line printers in the system.

nnn = zero to three-digit number indicating the numerical designation for a particular unit in a class of devices, such as "DTA010:" for DECtape drive Number 010.

: = a colon, an integral and terminating part of the device name.

If there is only one device in a particular class, the part "nnn" may be omitted. For example, "PLT:" is used to represent the system plotter, "PRG:" the Program Library, and "SYS:" the System Library. It may also be omitted in a multi-unit device name if only one such unit is assigned and available for general usage, such as "DSK:". A list of physical names of selected system devices is shown in Table 7.1.

Device	Physical Names
Array processor unit	APU:
Card Punch	CDP:
Card Reader	CDR:
DECtape Drive	DTA: DTA010:, DTA011:, etc.
Disk	DSK:
Line Printer	LPT: LPT03:, LPT06:, LPT10:, etc.
Magtape Drives	MT7: (7-track drive) MT8: (9-track drive, 800/1600 BPI) MT9: (9-track drive, 1600/6250 BPI) MTA010:, MTA011:, etc.
Operator's Terminal	CPR: or TTY0:
Plotter	PLT:, PLT010:
System Library	SYS:
Program Library	PRG:
Engineering Library	ENG:
Terminals	TTY: TTY0:, TTY16:, TTY63:, etc.

Table 7.1 System Devices and Their Physical Names

(2) Logical device name The user may also define the device with a name of his choice by the monitor command ASSIGN or MOUNT (see Chapter 8). Such a user-chosen name is called the logical device name. Once a logical name is assigned, a device may be referred to by either its physical or logical name. In case there is a conflict between a logical name and a physical name, the conflict may be resolved by the System which gives the logical name precedence. The format of a logical device name is:

LOGDEV:

where LOGDEV is an one to six-character alphanumeric string, and the colon is an integral and terminating part of the name.

(3) PIP switch In using the PIP program, all tasks are interpreted in terms of file transfer with a single command format. Variations of the tasks may be designated in the command structure by adding switches. There are two acceptable forms of PIP switches: a code letter preceded by a slash or a code letter enclosed in parentheses -- for example, /X or (X). Multiple switches may also be given in either form -- for example, /B/X or (BX). A PIP switch may be placed anywhere in a PIP command.

(4) File specification For the purpose of identifying a file, each file is given a name. Once the names are established, the system will maintain a directory so that users need not be concerned with the exact location on the disk for their files. For the DEC System-10, the format of a complete file specification is:

DEV: NAME.EXT [m,n] <xyz>

where: DEV: = name of device on which the file is stored. If this part is omitted in the complete specification, DSK: (the disk assigned to users) is assumed.

NAME = filename consisting of one to six alphanumeric characters with no embedded blanks.

.EXT = file extension consisting of zero (0) to three alphanumeric characters with no embedded blanks. If it contains zero character, it is called a null extension. The period is an integral part of the extension.

[m,n] = the PPN of the person who created and owns the file. the default PPN is the current job's PPN. Note the use of square brackets.

<xyz> = a three-digit protection code. Note the use of angular brackets.

The file extension is a part of file identification, used to indicate the type or language of the file. Although any zero to three-character combination can be used as a file extension for any file, the following are some of the most frequently used file extensions, and their meanings are recognized by the System.

Examples:

NEWTON.PIL	A PIL program file named NEWTON.
NEWTON.FOR	A FORTRAN program file named NEWTON.
NEWTON.REL	An object program compiled from NEWTON.FOR
NEWTON.BAS	A BASIC program file named NEWTON.
FOR01.DAT	A data file named FOR01.

Symbols "*" and "?" are used as "wild cards" to represent a class of file names or extensions, as illustrated by the following examples:

Examples:

NEWTON.*	All files named NEWTON of any extension.
*.FOR	All FORTRAN files.
.	All files.
F????.DAT	All data files whose names are 5 characters or less and begin with F.
D12??.D??	All files whose names begin with "D12" and

	contain 5 or less characters, and whose extensions begin with the letter D and contain 3 or less characters.
D12??.*	All files whose names begin with "D12" and contain 5 characters or less.

The protection code is a 3-digit octal number xyz, each digit ranging from 0 to 7. Each digit defines a protection level of the file against a certain class of users:

x = protection level against the file owner himself.

y = protection level against users sharing the same project number.

z = protection level against the general public.

The level of protection ranges from level 0 to 7, and level-7 is the highest. The exact definition of each protection level is given below:

Code Digit	Access Protection*
7	No access privileges
6	Execute only
5	Level 6 + Read privilege
4	Level 5 + append privilege
3	Level 4 + update privilege
2	Level 3 + write privilege
1	Level 2 + rename privilege
0	Level 1 + change protection privilege

Access protection can be changed by executing RENAME or PROTECT monitor command (see Chapter 6) or the PIP program (see Chapter 7). Since there are 8 levels of protection in each of three classes of users, there are 512 different shades of protection-level combinations possible. Normally, one need only be concerned with a few commonly used codes:

Protection Codes	Applications
077,177	Strictly private and non-sharable, such as grade files maintained by an instructor.
057;177	Sharable within a project, for example, a program to be shared by all students in a course.
055,155	Sharable with the computer community, but the file may not be modified by anyone except the file owner.

The System assigns a default protection level of 057, set automatically by the computer if the person does not specify any protection code when he creates the file. In some coursework, instructors may arrange to have the default protection level automatically set at 077. In such a case, the

---

*Subject to minor local variations. For example, at the University of Pittsburgh, access protection designated by the x-digit has been modified slightly.

protection code of a student's file is 077 to his classmates, but is 057 to his instructor.

(5) Directory and non-directory devices

While the PIP program deals mainly with transfer of files, many transfers are really input/output operations. For example, transferring a file from the disk to a line printer is actually an output operation of printing out a disk file. Therefore, if we ignore these input/output operations, we need only consider three major devices for the "true" transfer. They are the disk, the DECtapes, and the magnetic tapes.

For the purpose of file references, the disk and the DECtapes are called directory devices and the magnetic tapes are non-directory devices. In a directory device, the files are identified by their file specifications, and there is no need to know the actual physical locations of these files on the devices. However, on a non-directory device, a file can only be identified and located by its physical location or sequence, for example, file No. 3.

7.2 The Standard PIP Command Structure

The PIP program may be called at the monitor level by the command:

```
.R PIP
```

After PIP is loaded, a prompt character "*" is printed at the terminal, and PIP is ready to accept PIP commands.

The PIP command has a standard and simple structure:

```
Destination = Source(s)
```

where: Destination = device and/or file which is to receive the transferred data. This portion contains only one file specification. (Note: One file specification may imply multiple files if a wild card is used.)

"=" = an equal sign separating the source and the destination. It may be substituted by a left arrow available on some terminals.

Source(s) = one or more file specifications of the origin(s) of transfer.

Both the destination and the source specification are of the standard form *DEV: NAME.EXT[m,n]<xyz>*. Rules of default conditions and wild card construction apply as given in Section 6.1.

Exit from the PIP program to the monitor may be accomplished by pressing either the CTRL-C or CTRL-Z key.

Various uses of the PIP commands are shown in the following examples:

Example	Function
.R PIP	Call for PIP
**TTY:=CURVE.FOR*	Transfer a disk file to TTY. In other words, list the file on TTY.
(Program listing follows here.)	
*	Transfer completed; PIP is ready for another command.
*↑Z	Exit from PIP.

The whole sequnece is equivalent to issuing a monitor command *TYPE CURVE.FOR*. In fact, the monitor TYPE command actually activates and runs the PIP program.

Other examples and explanations are given below:

Example: **LPT: = CURVE.FOR*
Function: List the file CURVE.FOR on the line printer.

Example: **LPT: = *.FOR*
Function: List on the line printer (of the station where the terminal of the user belongs) all FORTRAN-10 files stored on disk.

Example: **DOUBLE.FOR = CURVE.FOR*
Function: Copy the disk file CURVE.FOR and name the copy as DOUBLE.FOR.

Example:
```
.DRIVE DTA
.MOUNT DTA:T1/WE/VID:A1004
.R PIP
*T1:DOUBLE.FOR = CURVE.FOR[115103,320571]
*↑Z
.DISMOUNT T1
.UNDRIVE
```
Function: Copy a file CURVE.FOR that belongs to another PPN onto a DECtape registered under the number A1004. the name of copied file on the DECtape is DOUBLE.FOR. Message printouts are not shown.

Example: **LPT: = MT9:**
Function: Print everything on the magnetic tape currently mounted on the tape drive.

Example:
```
.R PIP

*DSK: SAMPLT.FOR = TTY:

      A=1.2345
      WRITE(6,10)A
   10 FORMAT(F10.5)
      END              This CTRL-Z is an end-of-file mark,
↑Z                     therefore terminates TTY input. The
*↑Z                     second CTRL-Z is to exit from PIP.
```
Function: Use PIP to enter a source program. A possible but not a recommended way, because no editing is possible.

7.3 Transfer of Multiple Files, the X-Switch

When there is more than one file in the source specification, transfer may be done in two ways: to transfer collectively as one combined file, or to transfer files singly and retain individual identifications. In the former case, it is a natural application for merging several files together. In the latter case, an X-switch in the command will make the transfer of multiple files singly and each transferred file will be given a unique name, as specified in the command string.

File transfer between one directory device to another is a simple matter since both the destination and the sources are identified by file specifications. File transfer between non-directory and directory devices are more complicated since the filenames must sometimes be arbitrarily generated.

The examples below demonstrate many salient points of multiple-file transfer, with and without the X-switch:

Example: **DTA1: /X = DSK: *.* or DTA1:/X=*
Function: These two commands perform the same way. They transfer all disk files to DTA010 and retain individual filenames and extensions. When the right side of the equal sign is blanked, the default conditions simply mean "everything from my disk".

Example: **DTA1: MESS.MT9 = MT9:**
Function: Transfer all files combined on the magtape MT9 to a DECtape. Name the result MESS.MT9. Since no X-switch is used, all files are merged into a single file after the transfer.

Example: **DTA1: MESS.MT9/X = MT9:**
Function: Transfer all files from MT9 to DTA010, retaining the individual filename and extension of each file. However, since only one filename and one extension are specified on the destination side, individual filenames and extensions must be generated by PIP. The rule of filename generation is as follows: The format of the generated filename is XYZnnn where:

XYZ = the first three characters in the specified destination filename. If none is specified, the three-character chosen is XXX.

nnn = a three-digit number, from 001 to 999.

The extension is retained and shared by all generated file specifications. Thus, for this example, if MT9 contains 15 files, the transferred files on DTA010 will have names of MES001.MT9, ..., MES015.MT9.

7.4 Transfer of Files with Editing

Certain editing functions can be incorporated into the transfer of files. A list of selected switches for this purpose is shown in Table 7.2 on the next page. The carriage control characters created by the /P switch will instruct a line printer to perform certain actions as listed in Table 7.3:

Switch	Function
To generate or delete sequence numbers.	
/S	To insert sequence numbers. At the start of each line of a file, a sequence number is computed and inserted. These sequence numbers assigned by PIP are five-digit numbers starting from 00010 and ranging through 99990 in increments of 10. Example: Source file: Destination file after /S switch: A=1.2345 00010 A=1.2345 WRITE(6,10) 00020 WRITE(6,10) 10 FORMAT(F10.5) 00030 10 FORMAT(F10.5) END 00040 END
/0	Same as /S, except that the increment is 1.
/N	To delete sequence numbers from the file.
/E	To replace characters in columns 73-80 in each line by spaces, and to ignore the sequence number on each line. The E-switch may be used for any input device but is most commonly used in a card reader.
To delete trailing spaces on each line to save storage space:	
/C	To delete trailing spaces in each input line and to convert multiple spaces into tabs. Its main purpose is to conserve storage space by making the file more compact.
/T	To delete trailing spaces on each line. No conversion of multiple spaces to tabs.
To perform "line-blocking" of input data files:	
/A	To "line-block" the file so that each buffer contains an integral number of lines, and no lines are split between physical output buffers. Such splitting may cause unpredictable read-errors. This is a necessary step if the input data files, prepared by an editor such as the UPDATE, are for a FORTRAN F40 program. For FORTRAN-10 (F10), line-blocking of input data files is no longer necessary.
To prepare a FORTRAN output file for printer output:	
/P	To convert a FORTRAN output file containing printer control characters into one that will activate the carriage control of the line printer. Without such conversion, the printer will simply print out the control characters as characters without taking any action. The FORTRAN carriage control character interpretation is shown on Table 7.3.

Table 7.2 PIP Switches for Transfer of Files with Editing

Carriage Control Character	Line Printer Action
blank	Normal single space printing. FORM FEED (advance to a new page) every 60 lines.
*	Normal single space printing. No FORM FEED, even when the bottom of a page is reached. This is used when continuous printing is desired, such as in a chart, a graph, or a long tabulation.
+	To overprint the previous line, such as to underscore part of the text.
,(comma)	To skip to the next 1/30 of page.
.(period)	To skip to the next 1/20 of page.
/	To skip to the next 1/6 of page.
=	To skip two lines (triple space)
0	To skip one line (double space)
1	To skip to the top of the next page (FORM FEED)
2	To skip to the next 1/2 page
3	To skip to the next 1/3 page

Table 7.3 FORTRAN Carriage Control Characters

For example, a file containing the following:

```
(Column 1)
    1
    2
                    REPORT TITLE
    1
```

will print out the "REPORT TITLE" at the middle of the page, and the subsequent materials will begin at the next page.

7.5 File Directory Management

PIP switches in this group will handle directory management, such as reporting on the directory content, changing the file specifications, file deletions, etc. Althought the PIP command still conforms with the general format of source and destination, no actual transfer takes place. Switches in this group are listed in Table 7.4 on the next page.

7.6 Multiple PIP Switches

More than one switch may be given in a single PIP command to get a cumulative effect. This is illustrated by the examples below:

Example: **DTA1: (ZDX) = *.LST, *.LPT*
Function: This command is the same as the three successive PIP commands:
**DTA1:/Z =*
DTA1:/X = DSK:*.
**DTA1:/D = *.LST, *.LPT*
The net result is: Clear the DECtape now mounted on DTA010 and copy everything from the disk onto DTA010 except *.LST and *.LPT.

Example: **DSK:NEWTON.REL/B/P = DTA1:NEWTON.REL*
Function: This command will copy a FORTRAN binary file from a DECtape onto a disk in order to insert a control word into each physical buffer. If buffer sizes are the same, the P-switch is not needed. Also, FORTRAN-10 binary files need no P-switch.

Switch	Function
To list the directory:	
/L	To list the directory of the source devices and files. The directory will include filenames, extensions, protection codes, number of blocks, creation dates and total blocks. Examples: — Explanations: *TTY: = DSK:/L — List the disk file directory on TTY. *TTY: = DTA010:/L — List DTA010 directory on TTY. *TTY: = *.FOR/L — List all disk FORTRAN files on TTY. *LPT: = *.PIL, *.FOR/L — List directory on line printer. *LIST.DIR=DSK:*.*/L — Store directory as a file named.
/F	Fast listing of directory giving filenames and extensions only.
To change the file specifications:	
/R	To rename the source file(s) in the manner indicated in the destination file specification. Only one or one class of files (wild card construction) may be renamed in one PIP command. Examples: — Explanations: *NEW.FOR<155> = OLD.BAK — Straight copying of a file; OLD.BAK retained after copying. *NEW.FOR<155>/R = OLD.BAK — Rename; OLD.BAK no longer exists after renaming. *NEW.*/R = OLD.* — Change all filenames of OLD to NEW, and retain all extensions. **.*<177>/R = — Change the protection codes of disk files to 177.
To delete files:	
/D	To delete the specified source file from the destination device. Only one source device is permitted, and it is initially assumed to be the same as the destination device. Examples: — Explanations: *DSK:/D = OLD.FOR, *.REL — Delete from the disk all REL files and the file OLD.FOR. *T1:/D = T1: FILE1.DAT — Delete a file named FILE1.DAT from a DECtape which has been mounted and given a logical name of T1.
/Z	To zero out (erase) the directory of the output device. If the output device is the disk, an attempt is made to delete all the files whose names are found in the directory specified. If it is not possible to delete some of the files, the request will be terminated after as many files as possible have been deleted. Example: — Explanation: *DTA011:/Z = — Clear the DECtape mounted on the drive DTA011.

Table 7.4 PIP Switches for Directory Management

7.7 A Summary of PIP Switches

A summary of the selected PIP switches is given in Table 7.5 below:

Switch	Function
A	Line blocking.
B	Binary processing.
C	Suppress trailing blanks, convert multiple spaces to tabs.
D	Delete file.
E	Treat (card) columns 73-80 as spaces.
F	Fast listing of directory.
N	Delete sequence numbers.
O	Same as /S switch, except increment = 1.
P	FORTRAN output assumed. Convert format control characters for LPT listing. Use /B/P for copying binary files.
R	Rename file.
S	Sequence the file with sequence numbers, increment = 10.
T	Suppress trailing blanks.
W	Convert tabs to multiple spaces.
X	Copy specified files.
Z	Zero out the directory.

Table 7.5 A Summary of Selected PIP Switches

SORT

7.8 The SORT Program

SORT is a "stand-alone" program from the COBOL processor. It is used to sort a file according to the contents in a specified field of each record. The sorting may be done either numerically or alphabetically, in ascending or descending order. Since the program was developed as a part of the COBOL processor, many of its features are COBOL-related. Here, only a simplified version will be presented so that the application is confined to ASCII-coded and non-COBOL files. We shall see that even with such a restriction, there is a wide range of applications. The SORT program may be called at the monitor level by a command:

```
R SORT
```

when the terminal types out a prompt symbol "*", sorting commands may be issued. There is only one command format:

```
OUTPUT FILE SPEC = INPUT FILE SPEC/switch R/switch K
```

where OUTPUT FILE SPEC = file specification of the sorted result.

INPUT FILE SPEC = file specification of the original file.

Switch R, Switch K = two of many SORT switches available in the COBOL processor. They are defined in the following way:

(1) The R-Switch

The RECORD or R-Switch defines the length of each record and has a format of:

/RECORD:n or /R:n

where n=record length in number of columns. If "n" specified is smaller than the actual record length, the columns beyond the nth column will be deleted in the result. This switch must be given, and its omission is an error.

(2) The K-Switch

The KEY or K-switch defines the field in each record about which the file is to be sorted. It has a general format of:

/KEY:begin:size:order or /K:b:s:o

where: begin = an integer, representing the beginning column of the sorted field.

size = an integer, representing the size (number of columns) for the sorted field.

order = a character, "A" (for ascending order) or "D" (for descending order). If the order is ascending, the "order" parameter may be omitted in the key.

Any number of /KEY switches may be given. If there is more than one K-switch, then the first one is the primary sort key, followed by the secondary sort key, the tertiary sort key, etc. For example, suppose we sort a student roster in Engineering first by departments, then in each department by last names, and then by initials if last names are the same. Here we use three K-switches: the primary sort for the department, the secondary sort for the last names, and the tertiary sort for the initials. In the SORT command string, the leftmost K-switch is the primary key, and the key-hierarchy descends as you move toward the right.

Example: A grade file GRADE.DAT is stored on disk, and its content is:

```
          1         2         3
12345678901234567890123456789 0     (Column Numbers)

    ABBOT, W. E.     67
    DOE, J. Q.       75
    QUINCY, T. C.    83
    RIM, E. D.       47
    TIMMONS, E. E.   66
    YANG, R. Y.      88
```

```
.R SORT                                        Comments

*G1.DAT = GRADE.DAT/D:22:2/R:23          Sort GRADE.DAT according to
                                         descending order of grades.
(Computer run message)
```

Example: Suppose ROSTER.DAT is a student roster file. Each record is of 80-column wide. Information contents are stored in the following columns:

Columns 1- 3	Initials
Columns 6-20	Last names
Columns 21-25	Department names abbreviated
Columns 26-30	School names abbreviated

The following SORT command will sort the file: first according to the school, then within the same school according to the departments, then within the same department according to the alphabetic order of last names, and if the last names are the same, according to the alphabetic order of their initials:

```
*ROSTER.DAT = ROSTER.DAT/K:26:5/K:21:5/K:6:15/K:1:3/R:80
```

RUNOFF

INTRODUCTION

RUNOFF is a utility program which facilitates the word-processing applications on the DEC-10, such as for preparing manuals, reports, theses and dissertations, etc. The general procedure is as follows:

(1) A file is prepared by the user that contains text materials.

(2) Interspersed in the text file are appropriate RUNOFF commands that specify the case and formatting instructions. The file containing both text and RUNOFF commands is called a RUNOFF source file. For experienced RUNOFF users who are also their own typists, these first two steps are often merged into one.

(3) When such a source file is run, RUNOFF will take the file and reproduce it on the line printer, on a terminal, or into another file. In so doing, it also performs the formatting and case shifting as directed. If specified, it will also perform margin changes, line justification, page numbering, titling of each page, compiling index terms, etc.

The following example shows some typical results of RUNOFF.

CHAPTER 1

INTRODUCTION

1.1 Batch Processing versus Time-Sharing

Once upon a time, when a computer user wanted to run a program, he went through the following steps:

(1) The user submitted his program and data deck to the Computer Center.

(2) The decks of cards submitted by different users were stacked together to form a batch, each deck with its proper identification. All jobs in one batch were then executed in one "run", hence the name "batch processing". The information on the punched cards in a batch were first copied into a reel of magnetic tape by means of a small and relatively inexpensive computer. The reason for this was that the card-input to the main computer was a slow and therefore expensive process.

(3) The magnetic tape so prepared became the input medium to the main computer. At the scheduled time, the jobs in the batch were run and the outputs (printouts, cards, tapes, etc.) were obtained. Sometimes the outputs were recorded on another reel of magnetic tape; then output printing may be done off-line so as not to slow down the computer operation.

The chief benefit of RUNOFF is that the source file may be easily edited and modified by a text editor. Materials may be deleted or added. Formatting rules may be changed regarding margins and spacing. These changes normally result in a catastrophe dreaded by every typist and student because the material must be re-typed and re-paged. Now, the RUNOFF program is simply rerun with the revised source file, and a new copy is obtained properly revised and paginated. Thus, documentations, theses and dissertations may be updated and revised as necessary without requiring extensive re-typing.

7.9 RUNOFF Operating Procedure

RUNOFF can be called by a monitor command:

```
.R RUNOFF
```

when the terminal types out a prompting "*" symbol, RUNOFF is ready to accept a command. The general format of a RUNOFF command is:

```
Output specification = Input specification/switches
```

where the output specification may be a standard DEC-10 name for one of the following:

Line printer	such as LPT10:
Terminal	such as TTY:
Another file	such as THESIS.DOC

Example: The source file that contains a textual manuscript and RUNOFF commands has been prepared and stored as MAN.RNO in the user's disk. "RNO" is the default extension of a RUNOFF source file.

Commands	Comments
.r runoff	Call for RUNOFF.
*LPT10:=MAN	To produce a copy of finished manuscript on the line printer No. 10
*TTY:=MAN/PAUSE	To produce a copy of manuscript on the user's terminal. Pause at the beginning of each page to allow alignment of paper.
*MAN.DOC=MAN.RNO	To produce a file MAN.DOC that is a copy of the finished manuscript.
*MAN	To produce a file MAN.MEM from a RUNOFF source file MAN.RNO. Here file names are the same and extensions are default extensions.

Several operating hints may be useful:

(1) If the manuscript is reproduced on a terminal, make sure the terminal settings are adequate. The terminal is normally set at a set of default values, such as tabs at 9,17,25..., right margin at 72, characters at upper cases, etc. If the manuscript requires non-standard settings, appropriate commands must be given to pre-set the terminal. The following are some typical and useful TTY commands applied before a RUNOFF session:

.TTY WIDTH 132	Set right margin at 132. Right margin can be set at anywhere between 17 and 200.
.TTY PAGE	This will enable the control functions of CTRL-S and CTRL-Q keys. CTRL-S will suspend the output (but not kill it), and CTRL-Q will resume it. Suspension of output gives you a chance to inspect the output. This is very important if you are using a CRT terminal.
.TTY LC	Normally your terminal is set for upper case, even if it has lower case capability. To prepare a source file in both upper and lower cases, the lower case must be activated.

(2) If the manuscript contains upper and lower cases, find out which printer in the System has a lower case capability. Otherwise, all lower cases will be forced into upper cases, a situation that may be objectionable on occasion.

A RUNOFF PRIMER

7.10 How RUNOFF Works

The complete RUNOFF utility on the DEC-10 contains between 80-90 commands altogether. However, only a handful is in frequent use and is therefore essential. Hence, it is possible and advisable for a beginning user in RUNOFF to master this small set so as to quickly utilize the RUNOFF capability.

When the RUNOFF is called, certain modes and formatting instructions are already set up, and these are the standard default RUNOFF status:

(1) Print page numbers on every page except the first.

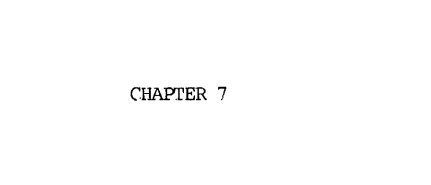
(2) Single space with left margin set at 0, right margin at 60. This means that all text reproduced by RUNOFF will be left-justified at column-1 and right-justified at column-60.

(3) Paper size is assumed to be 60-character wide and 58-lines long.

(4) Tab stops are set at DEC-10 default values, namely, at 9, 17, 25, 33, 41, 49, 57, 65, etc.

As the RUNOFF proceeds to reproduce the source file, it constructs a line by copying words and leaving one blank between words, two blanks after a period, until adding another word would overshoot the right margin. This process is called filling. After a line is filled, blanks are added as necessary between words on that line until the last character on that line is aligned with the right margin. This process is called justifying.

Therefore, if there is no RUNOFF command at all in a textual file, RUNOFF will reproduce it into a document, using only default formats and modes. The document produced will have an appearance as shown below:



The RUNOFF detects the entity of a word by marking off word-delimiters. A word is delimited in RUNOFF by one of the following: space(s), tab(s), linefeed or carriage return. Multiple and consecutive delimiters, such as multiple spaces, count as a single delimiter. Thus the source file can be prepared without worrying about the width of each line and the number of spaces between words, because RUNOFF will now treat a carriage return between two words and spaces between two words the same way. One exception is when a carriage return is followed by a space:---that is, when a line begins with a space at column 1. When the "automatic paragraph" mode is on, a space at column 1 is regarded as the beginning of a new paragraph, and RUNOFF initiates a specified format for a new paragraph, such as line spacings and indentation. On the other hand, punctuation marks such as ",", "?", etc are not recognized as word delimiters. Thus, the text "delimiters,and" is treated as one word, while "delimiters, and" is treated as two words. If you have a typing habit of not leaving a space after a punctuation mark, beware.

The use of the terms "space", "vertical space", and "line space" is likely to be ambiguous and confusing, and needs to be clarified for the subsequent presentation in this chapter.

The term space, when used alone, will denote the horizontal space on a line. Most terminals and printers are set at a horizontal scale of 10 character/inch, and a space will measure 1/10 inch. For terminals with "Elite" type, each space measures 1/12 inch. The term "vertical space" typically measures 1/6 inch in the vertical direction of typing. Terms such as "single-space" and "double-space" refer to the vertical spacings. The term "line space" corresponds to the spaces between lines. Thus, if the default page size is 60 horizontal-spacings by 58 vertical-spacings, there is room for 58 line spacings for single-spacing output, but only 29 line spacings for double-spacing output. With these definitions, there could be a difference in saying "skip 2 lines" versus saying "skip 2 vertical spaces".

7.11 Basic RUNOFF Commands

Interspersed in the text file are the RUNOFF commands which are identified by a period "." in column 1. Thus, to avoid misinterpretation by RUNOFF, the text itself should not permit a period in column 1.

There are four types of RUNOFF commands according to their functions. A complete summary of the RUNOFF commands in these four classifications will be deferred to Section 7.14. Here, a limited set of basic commands will be presented. Each command is presented in its complete form and its abbreviation, and the lower case part of each command denotes an argument or a parameter.

Before getting to the basic set of RUNOFF commands, one important implicit command will be explained first. This command is rarely used as an explicit command in the source file, yet it is often built into other commands. Thus its action is often implied and included.

.BREAK .BR	This command will cause a BREAK, i.e. the current line will be output with no justification, and the next word of the source text will be placed at the beginning of the next line.

(1) To set margins, spacings, and page size

These are commands to set the left and right margins, vertical spacings, and the page size to values other than the current ones.

.LEFT MARGIN n .LM n	Set the left margin to n. "n" must be less than the right margin but not less than 0. Default=0.
.RIGHT MARGIN n .RM n	Set the right margin to n. "n" must be greater than the left margin. The default setting is 60.
.PAPER SIZE n,m .PAGE SIZE n,m .PS n,m	Set the size of page n lines by m columns. The default setting is 58,60.
.SPACING n .SP n	Set a ratio between vertical spacing and line spacing. The n can be from 1 to 5. The default setting is 1. ".SP 1" is for single-spacing; ".SP 2" is for double-spacing.

Examples: The following shows the commands in the source file and their effect on the output. The RUNOFF commands are highlighted in italics:

Source File

```
.LM0 .RM20
X XX XXX XXXX XXXXX XXXXXX XXXXXX
XXXXX XXXXX XXXX XXXX XXX XXX XX.
.LM8
YYYYY YYYYY YYYY YYYY YYY YYY,
YY YY Y.
.LM5 .RM15
ZZZ ZZ ZZZZZ ZZZ ZZ
ZZ ZZZ.
```

Output

```
X XX XXX XXXX  XXXXX
XXXXXX  XXXXXX XXXXX
XXXXX XXXX XXXX  XXX
XXX XX.
       YYYYY  YYYYY
       YYYY    YYYY
       YYY YYY,  YY
       YY Y.
   ZZZ     ZZ
   ZZZZZ  ZZZ
   ZZ ZZ ZZZ.
```

(2) To set text format

The following commands set the text paragraphing formats:

.AUTOPARAGRAPH
.AP

This command causes a blank line or any line starting with a space to be considered as the start of a new paragraph. Format of a new paragraph is specified by a specified or a default ".PARAGRAPH" command.

.NOAUTOPARAGRAPH
.NAP

This command cancels the AUTOPARAGRAPH mode.

.PARAGRAPH n,v,t
.P n,v,t

This command causes a BREAK action and takes the next line as a new line and the beginning of a paragraph. In the meantime, the formats of the subsequent paragraphs are set by the parameters n, v, t:

n = number of indented spaces. The default value is 5, and n can also be negative. Negative indentation means a paragraph beginning to the left of the left margin of paragraph, such as this paragraph here.

v = number of line spaces between paragraphs. It can range from 0 to 5.

t = argument in the .TEST PARAGRAPH t command, which is automatically executed when paragraphing.

.BLANK n
.B n

This command will cause a BREAK action and insert n blank line spaces after the last line. The parameter n can be negative to move the line to n lines from the end of the page. Note that the actual number of blank vertical spaces is equal to (n*spacing per line).

.CENTER n;text
.C n;text

This command will cause a BREAK and center the "text" in the source file. The centering is over column n/2, independent of the setting of the left and right margins. If n is not given, it is assumed to be the page width.

Example: The following shows the paragraphing control:

Source File

```
.LM0 .RM20
.AP
.P3,1,1
.C20;EXAMPLE
 X XX XXX XXXX XXXXX XXXXXX XXXXXX
XXXXX XXXXX XXXX XXXX XXX XXX XX.
 YYYYY YYYYY YYYY YYYY YYY YYY,
YY YY Y.
.LM5 .P-3,1,1
 ZZZ ZZ ZZZZZ ZZZ ZZ
ZZ ZZZZ ZZZ ZZZZ
ZZZZZZ ZZZ.
```

Output

```
                EXAMPLE

   X  XX  XXX   XXXX
XXXXX  XXXXXX XXXXXX
XXXXX   XXXXX   XXXX
XXXX XXX XXX XX.

   YYYYY YYYYY  YYYY
YYYY  YYY YYY, YY YY
Y.

  ZZZ ZZ  ZZZZZ  ZZZ
     ZZ  ZZ ZZZZ ZZZ
     ZZZZ     ZZZZZZ
     ZZZ.
```

Note that if in the above example, paragraphing were to go back to ".LM0.P3,1,1", the RUNOFF command should be ".P3,1,1 .LM0" rather than ".LM0 .P3,1,1". At the end of the above example, the paragraphing is set at negative indentation of 3 spaces. Thus if ".LM0" is applied while negative indentation is still in effect, you will be telling RUNOFF that the paragraph should begin to the left of column 1!

(3) To control upper and lower cases

.UPPER CASE .UC	This command sets the output mode to upper case. All alphabets in the source file, upper or lower cases, will be forced into upper case in the output.
.LOWER CASE .LC	This command sets the output mode to lower case. All alphabets in the source file, whether upper or lower cases, will be forced into lower cases in the output.

If .UC and .LC commands are both applied, the text cases will be reproduced as they are in the source file.

To exert control of upper or lower case on individual characters, RUNOFF uses special symbols. This is discussed in Section 7.12.

(4) To control filling and justifying

RUNOFF assumes that the source file is to be filled and justified, and hence considers them default actions. However, on occasion, you may wish to reproduce the text exactly as given -- for example, a tabulation by columns. Therefore, it is necessary that you should be able to enable and disable the filling and justification process by commands. They are listed below:

.FILL .F	This command sets RUNOFF to a filling mode to add successive words from the source text until adding one more word will exceed the right margin. It also sets the justification mode specified by the last appearance of a JUSTIFY or NOJUSTIFY command.
.NO FILL .NF	This command will disengage the FILL and the JUSTIFY modes.
.JUSTIFY .J	This command sets the JUSTIFY mode that will increase spaces between words until the last word exactly meets the right margin.
.NOJUSTIFY .NJ	This command turns off the JUSTIFY mode, but the FILL mode is unchanged. An output line is filled but not justified, giving a ragged right margin.

The FILL and the JUSTIFY modes and their disabling commands have interacting influences. Turning off FILL would also turn off JUSTIFY, but turning off JUSTIFY does not affect the FILL. Because of this, note the following:

a. The NOFILL-NOJUSTIFY mode need be used only where there are several lines of material to be copied exactly. If there is only one line, it is not necessary to use this mode if there is a BREAK before and after the line.

b. Normally, FILL and NOFILL are used to turn both filling and justification on and off. It is usually desirable to do both. A subsequent appearance of a justification command will override the fill command.

c. A combination of FILL and NOJUSTIFY commands will produce a ragged right margin.

d. A combination of NOFILL-JUSTIFY will expand each line to justify, but this mode has doubtful utility. If this is applied, a JUSTIFY command should be given after every NOFILL command.

Example: The following example shows the difference between a justified and a ragged-right margin:

Original Text	RUNOFF Output: FILL and JUSTIFY	RUNOFF Output: FILL only
This command sets RUNOFF to a FILL mode to add successive words from the source text until the adding of one more word will exceed the right margin. It also sets the justification mode to be that specified by the last appearance of JUSTIFY or NOJUSTIFY command.	This command sets RUNOFF to a FILL mode to add successive words from the source text until the adding of one more word will exceed the right margin. It also sets the justification mode to be that specified by the last appearance of JUSTIFY or NOJUSTIFY command.	This command sets RUNOFF to a FILL mode to add successive words from the source text until the adding of one more word will exceed the right margin. It also sets the justification mode to be that specified by the last appearance of JUSTIFY or NOJUSTIFY command.

7.12 Special Text Characters

A number of text characters, when placed in conjuction with the text material, exercise a change of the case and the mode operations. They are listed below, with examples following:

(1) Ampersand (&) This character is used to specify underscoring of text. The character placed immediately after a "&" (no space in between) will be reproduced in the output as underscored. The underscoring can be turned on and locked by a double character "^&", and turned off and locked by "\&".

(2) Circumflex (^) and Back-slash (\) These two characters act as the up-shift and the down-shift keys of a typewriter. Thus, when a "^" is placed before a letter of the alphabet, that letter is reproduced in upper case. Similarly, a "\" in front of a letter of the alphabet causes the letter to be reproduced as a lower case. Also, "^^" will lock the shift key at upper case, and "\\" will lock the shift key at lower case.

(3) Number sign (#) Occasionally, it is necessary to include a fixed number of spaces in the text which should not be treated as word separaters. The character (#) is used by RUNOFF as a "quoted space" or a mandatory space that can neither be reduced nor expanded. Thus, if a fixed number of spaces is required in the output, those spaces may be reserved by a specified number of #'s. For example, "Section 2.13" may be reproduced as "Section 2.13", "Section 2.13", "Section 2.13", etc, depending on filling and justification of that line. These two words may even be split between two lines. However, "Section#2.13" can only be reproduced as "Section 2.13" on the same line.

(4) Less-than sign (<) If a RUNOFF command of "FLAG CAPITALIZE" has been given at the beginning of the source file, then a less-than sign (<) placed immediately before a word will capitalize that whole word.

(5) Greater-than sign (>) If a RUNOFF command of "FLAG INDEX" has been given at the beginning of the source file, then a greater-than sign (>) placed immediately before a word will place that word in the index buffer memory. At the end of the source file, a command of "DO INDEX" will produce an index section, where all indexed terms are sorted in alphabetical order and each with page reference.

(6) Exclamation mark (!) This character marks the beginning of comments or the end of footnote lines.

(7) Semicolon (;) This symbol indicates multiple commands.

(8) Underscore (_) The above listed special characters are recognized by RUNOFF for having certain special control functions. Hence, these special characters will not be reproduced in the output. If a special character is to be reproduced as it is, its control function may be temporarily disengaged by placing an underscore (_) in front of that character. For example, "A&P Stores" and "A_&P Stores" will be reproduced respectively as "A_P Stores" and "A&P Stores". The underscore sign can also be used to specify its own reproduction. For example, a (__) and a (____) text will be reproduced as (_) and (__) respectively in the output.

Examples: The following shows the functions of special characters and their effect on the output:

Special Character	RUNOFF Source	RUNOFF Output
&	&Nth derivative CT=&Computerized &Tomography	Nth derivative CT=Computerized Tomography
^&,\&	^&RUNOFF\&	RUNOFF
^ ^^	.LC ^UNITED ^STATES ^^UNITED STATES\\	United States UNITED STATES
\ \\	.UC AMERICA A\\MERICA	AMERICA America
#	T. W. Sze T.#W.#Sze	T. W. Sze T. W. Sze
_	_#Footnote: #Footnote:	#Footnote: Footnote:
<	.FLAG CAPITALIZE If number1<number2, If number1_<number2, If <number1_<<umber2,	If number1NUMBER2, If number1<number2, If NUMBER1<NUMBER2,

7.13 Selected RUNOFF Switches

The switch is an optional part of the RUNOFF execution command string:

Output Spec = Input Spec/Switches

The switches are indicators, each consisting of a slash and a keyword, plus an optional argument. These switches are used to set or select program options. Many switches perform similar functions as the RUNOFF commands for setting or selecting modes and formats. In these cases, a switch would have the same effect as including a RUNOFF command at the beginning of the source file. Thus, if the function of a switch is already specified by either the default conditions or by specific commands in the source file, that switch would be superfluous. Therefore, the usefulness of a switch is to augment a source file for those functions not specified or different from the default conditions. Therefore, the function of a switch should be regarded as setting an initial mode or condition. Whether that initial mode or condition will hold for the entire output depends on whether that mode or condition will be revised by commands downstream in the source file.

A selected group of RUNOFF switches are tabulated next:

Switches	Functions
/AUTOPARAGRAPH	A leading space in column 1 initiates format for a new paragraph.
/NOAUTOPARAGRAPH	Turn off auto-paragraph mode.
/CASE:LOWER /CASE:UPPER	Execution starts in lower case mode. Execution starts in upper case mode. UPPER is the default condition.
/DOWN:n	Move down text of each page by n lines. Default n=0.
/LINES:n	Initial page size in n lines. Default n=58.
/PAUSE	Pause between pages (to allow paper changes).
/RIGHT:n	Move to the right text of each page n spaces. Default n=0.
/SEQUENCE	List record number of source file at the left of the RUNOFF output.
/SIMULATE	Advance to the next page by form feeds.
/SPACING:n	Start with vertical spacing setting n. Default n=1.

7.14 A Summary of RUNOFF Commands

RUNOFF commands fall in four categories: (1) Text formatting, (2) Page formatting, (3) Mode setting, and (4) Parameter setting. They are listed below:

(1) Text formatting commands:

Command & Abbreviation	Argument	Function
.BLANK .B	n	To skip n lines
.BREAK .BR		To start a new line of output.
.CENTER .CENTRE .C	n	To center the next line around column n/2
.COMMENT	text	Comment, ignored by RUNOFF execution.
.FIGURE .FG	n	To reserve a space for an n-line figure.
.FIGURE DEFERRED	n	To defer an n-line figure to the next page if there is no room in the current page.
.FOOTNOTE .FN	n	To start an n-line footnote. (input footnote lines until "!" in col.1.)
.LIST .LS	n	To start a list of items with spacing n.
.LIST ELEMENT .LE		To start listing items.
.END LIST .ELS		To end a list.
.INDENT .I	n	To indent the next line n spaces.
.NOTE .NT	text	To start an indented note with the heading "text" centered.
.PARAGRAPH .P	n,v,t	To start a new paragraph. Equivalent to: .I n .S v .TP t
.SKIP .S	n	To skip n*spacing vertical spacings.

Table 7.6 RUNOFF Text Formatting Commands

(2) Page formatting commands:

Command & Abbreviation	Argument	Function
.APPENDIX (.AX)		Start next appendix with rest of line as name.
.CHAPTER (.CH)		Start a new chapter with rest of line as name.
.DO INDEX (.DX)		Output index with rest of line as title.
.END SUBPAGE .ES		Stop subpage numbering
.FIRST TITLE .FT		Include title on the first page.
.HEADER LEVEL .HL	n	Start section at level n(=1 to 5); the rest is the name.
.HEADER (.HD)	case	Issue "page" in case(=UPPER,LOWER,MIXED,or NONE)
.INDEX		Insert rest of this line in index.
.NO HEADER (.NHD) .NO NUMBER (.NNM) .NO PAGING (.NPA) .NO SUBTITLE (.NST)		Suppress page header. Suppress page numbering. Stop spliting into pages. Suppress subtitles.
.NUMBER APPENDIX	n	Set appendix to Appendix n.
.NUMBER CHAPTER	n	Set chapter number to n.
.NUMBER INDEX		Set chapter heading to "INDEX".
.NUMBER PAGE .NUMBER (.NM)	n	Resume page numbering at page n.
.NUMBER SUBPAGE	ch	Set subpage number to ch (A-z).
.PAGE (.PG)		Start a new page.
.PAGING (.PA)		Resume breaking into pages.
.PRINT INDEX .PX		Start printing the index.
.SUBPAGE (.SPG)		Start sub-page numbering.
.SUBTITLE (.ST)		Use the rest of the line as the subtitle.
.TEST PAGE .TP	n	Skip to a new page if fewer than n lines are left.
.TITLE (.T)		Use the rest of the line as the title.

Table 7.7 RUNOFF Page Formatting Commands

(3) Mode setting Commands:

oCommand & Abbreviation	Argument	Function
.FILL .F		Resume filling and justifying each line.
.NO FILL .NF		Stop filling and justifying.
.JUSTIFY .J		Resume justifying the text.
.NO JUSTIFYING .NJ		Stop justifying.
.LITERAL .LT		Treat the following lines exactly as they appear, including special characters.
.END LITERAL .EL		Terminate LITERAL treatment of text.
.LOWER CASE .LC		Lock in the lower case mode (°°).
.UPPER CASE .UC		Lock in the upper case mode (°°)
.PERIOD .PR		Leave two blanks after these punctuation marks (.!?:;).
.NO PERIOD .NPR		Stop the PERIOD mode.
.FLAG .FL	ch	Turn flag ON according to the flag argument ch:(CAPITALIZE,CONTROL,ENDFOOTNOTE,INDEX, LOWERCASE,QUOTE,SPACE,SUBINDEX,UNDERLINE, UPPERCASE).
.NO FLAG .NFL	ch	Turn off specified flag.
.FLAG ALL		Turn on all flags.
.NO FLAG ALL .NFL		Turn off all flags except (.I).

Table 7.8 RUNOFF Mode Setting Commands

(4) Parameter Setting Commands:

Command & Abbreviation	Argument	Function
.AUTOPARAGRAPH .AP		Accept line with leading spaces as the beginning of a new paragraph.
.NOAUTOPARAGRAPH .NAP		Deactivate AUTOPARAGRAPH.
.AUTOTABLE .AT		Accept line without leading space as the beginning of a new table.
.LEFT MARGIN .LM	n	Set left margin at nth column. Text begins on the n+1th column.
.RIGHT MARGIN .RM	n	Set right margin at nth column. Text stops at the nth column.
.PAGE SIZE .PS	n,m	Set the page size to n lines by m columns.
.SPACING .SP	n	Set vertical spacing between lines as n.
.STANDARD .SD	n	Set standard setup with width of n columns.

Table 7.9 RUNOFF Parameter Setting Commands

OPRSTK

7.15 Introduction

The DEC System-10 is so designed that a batch job may be submitted either in cards at a card reader or by a disk-stored control file at a remote terminal. In both cases, the output printouts are returned only at the operating station, usually at a line printer, and results of terminal-submitted batch processing jobs are not printed at the terminals unless they are done as described in Section 7.17. Terminal-submitted batch jobs have serious drawbacks. On one hand, the prime advantage of man-machine interaction is lost; on the other, the turn-around-time is not improved. In the meantime, it retains the high overhead of using the time-sharing system, and is therefore more expensive. However, on occasions, submitting batch jobs at a terminal may be the only way to get your job done. A user may be located at a remote place when batch jobs are to be submitted, and a terminal may be the only input device accessible to him. Or, since in general a time-sharing user has a smaller core allocation than a batch user, submitting his job as a batch job may be the only way he can run a large program.

DEC System-10 has monitor commands of SUBMIT and QUEUE INP: for submitting batch jobs at a terminal. However, at the University of Pittsburgh, these two commands are disabled to allow a unification of all queue processes, including the batch queue. A service program called OPRSTK (Operation Stacker) is implemented which replaces the DEC's CDRSTK (Card Reader Stacker) program.

To submit a batch job at a terminal, the following steps should be taken after the user has signed on at the terminal:

(1) Create a control file using an editor such as the UPDATE (see Chapter 2). Save the file on disk and give it a name with an extension of CTL, such as NAME.CTL.

(2) Run the OPRSTK program by either of the two monitor commands:

```
.R OPRSTK
ENTER FILE SPECIFICATION > NAME.CTL
```

or simply,

```
.OPRSTK NAME
```

After the job is queued, the system will respond with a verification message. Output results are printed at the system facilities, but no result will be available at the user's terminal.

The details of selected batch commands will be given in Chapter 9. We will only deal with the creation of a control file and its submission here.

7.16 To Create a Control File

Creation of a control file on disk by the UPDATE editor is no different from creating any other file. Only one point should be remembered. Such a control file will contain a number of BATCH commands which begin with a "$" in

column-1. When such a line is created by UPDATE, it will be misinterpreted by the editor as an UPDATE command. The problem may be solved either by starting the "$" at column-2 and once entered, removing the blank in column-1 by a $CHANGE command, or by using an UPDATE command of $IS # at the beginning of the editing session. In the latter case, only those lines beginning with "#" in column-1 will be taken as UPDATE commands.

Two examples will be given here, to be carried out from the creation of control files through their submission and execution as batch jobs. Both are examples used in this book---solution of a cubic equation by the Newton-Raphson method, one example by FORTRAN programming, and the other by PIL programming. We assume that these two programs have already been stored in the disk and named as NEWTON.FOR and NEWTON.PIL respectively, and their contents are:

NEWTON.FOR

```
      READ(5,10)A,B,C,D,X1
   10 FORMAT(F21.7)
    1 X2=X1-(A*X1**3+B*X1**2+C*X1+D)/(3.*A*X1**2+2.*B*X1+C)
      WRITE(6,10)X2
      IF (ABS((X1-X2)/X2-0.001)3,3,2
    2 X1=X2
      GO TO 1
    3 WRITE(6,11)X2
   11 FORMAT(/' THE REAL ROOT =', F20.7)
      STOP
      END
```

NEWTON.PIL

```
1.1    DEMAND A,B,C,D,X1
1.2    SET X2=X1-(A*X1**3+B*X1**2+C*X1+D)/(3*A*X1+2*B*X1+C)
1.3    TYPE X2
1.4    IF ABS OF ((X1-X2)/X2) $LE 0.001, TO STEP 1.6
1.5    SET X1=X2
1.51   TO STEP 1.2
1.6    TYPE IN FORM 1, X2
1.7    STOP
FORM 1.
 THE REAL ROOT = ---.----!!!!
```

The example was run with a given equation of:

Thus the input data are 1, -16, 65, -50 and 16 respectively for A,B,C,D,X1. The control files needed are:

FORTRAN Control File FORT.CTL

```
$JOB[115103,320571]
$DATA
1.
-16.
65.
-50.
16.
$EOD
.EXECUTE NEWTON.FOR
$EOJ
```

PIL Control File PIL.CTL

```
$JOB[115103,320571]
.PIL NEWTON
DO PART 1
1
-16
65
-50
16
$EOJ
```

7.17 To Submit a BATCH Job at a Terminal

Once the control file is prepared, submission of a batch job is simply to supply the control file name in running the service program OPRSTK. The proceedings are given next as illustrations:

Example: Control files have been prepared as FORT.CTL and PIL.CTL.

```
(LOGIN proceedings here)

.OPRSTK  PIL
$JOB[115103,320571]
;;; END OF JOB AFTER 10 CARDS /SEQUENCE NUMBER IS 7264 ;;;
EXIT
.OP FORT
$JOB[115103,320571]
;;; END OF JOB AFTER 10 CARDS / SEQUENCE NUMBER IS 7266 ;;;
```

The job logs of both jobs are included in Chapter 9, where the details of batch jobs will be given.

If a user wishes to capture the output data to type it out on his terminal, to print multiple copies, or to use it as an input in the subsequent processing, he may do the following:

```
.ASSIGN DSK: 6
.OPRSTK FORT.CTL

(Wait until the batch job is done.  Use QUEUE command
 to inquire about the job status.  See Chapter 8)

.TYPE FOR06.DAT
```

VIRTUAL MEMORY

7.18 The Virtual Memory Procedure

In a multi-programming and multi-processing system, it is often not possible to accommodate all the programs and data in the main memory at the same time. Those programs and data not being executed at the time are moved out of the memory and stored temporarily in the fast mass storage. Then as the user is assigned his time slice, his programs and data are moved back into the main memory. This is a standard and unique operation, called swapping, in a time-sharing system where the main memory is shared by many users and many programs. The mass storage in this case is called a swapping device.

The principal mechanism of the swapping process is the techique of memory map, which translates the addresses produced by the processor into addresses in the physical memory. With the memory map handling the address translation, the user is free from the task of keeping track of memory locations before and after swapping.

The main memory is allocated to the user in units of "Ks" or "Ps", where 1K memory is equal to 1024 words, and 1P (P for page) is 512 words. Each user is allocated a physical memory, whose size is limited by the user's memory limit or memory request, whichever is smaller. When the program and/or data are too large for the allocated size, the program will attempt to access memory outside the allocated area, causing a fatal memory access error called a "memory fault." The job is then aborted. In such a case, the user must scale down his problem or apply certain special techniques such as overlays and chaining of programs.

The techniques of swapping can now be extended to the user's program execution. A part of the user's program and/or data will be in his memory in the usual way; the rest is in the mass storage. When a "memory fault" occurs, the fault manager (a software unit) will take note of the fault. It will then bring into the main memory a page from the mass storage, and take out a page from the memory (back into the mass storage) to make room. Thus, a memory access fault may cause some time delay due to the swapping (called the overhead), but it will not cause the job to abort. Here from the view point of the user, the mass storage is *in effect*, although not *in fact*, a part of his memory allocation. Therefore, such a memory is then called the virtual memory. In contrast, the actual memory allocated to the user is the physical memory.

Assuming that the virtual memory will be used only in a batch job, the procedure of using the virtual memory technique is outlined as follows:

(1) The $JOB card should include a CORE switch specifying the core request. Without this switch, the system default allocation applies for the job.

(2) Set the physical core limit by a monitor command:

```
.SET PHYSICAL LIMIT mK
```

where *mk* is the core size in K-words. If this is omitted, the physical limit is assumed to be that requested in the $JOB card.

(3) Prepare a core image (a SAV or EXE file) of the programs and save it as a SAV or EXE file. This may be done by using the LINK-10 loader by either of

the two following ways*:

```
.LOAD/LINK list
.SAVE flname
```

or,

```
.LOAD/LINK   flname/SAVE, list
```

where *list* contains a list of programs to be loaded, and *flname* is the name of the EXE file to be saved. Note that the /LINK switch must be placed in front of the list of program names.

(4) Request virtual memory facility by setting its limit:

```
.SET VIRTUAL LIMIT nK
```

The virtual memory limit must not be more than twice the physical limit, or $n \leq 2m$. Also, after the current job is finished, the virtual limit should be reset back to zero.

(5) Run the program, using the EXE file just saved:

```
.RUN FLNAME
```

Although the virtual memory facilities are available, users should exercise considerable restraint in using them, because they expand the memory at a great expense to efficiency. Some installations make a policy of using the virtual memory as a last resort. When a user does use the technique, he should examine his program very carefully and modify it if necessary. The general principle is that the execution should have access to contiguously stored data or programmed steps. For example, the order of subscripts as indexes becomes critically important in a multiple-dimension array processing. Also, the program should try to avoid branching statements, such as GO TO. In any event, this facility is available only by special permission or arragement.

---

*If the LINK loader is the default loader of the system, the "/LINK" switch shown below will not be needed.

REFERENCES

1. DEC SYSTEM-10 UTILITY MANUAL, DEC-10-UTILA-A-D, Digital Equipment Corporation, Maynard, Massachusetts; 1975

2. SORT AND CSORT, DEC-10 Notes, Computer Center, University of Pittsburgh, Pittsburgh, Pennsylvania; June, 1979.

3. DEC System-10 SORT/MERGE USER'S GUIDE, DEC-AA-0997D-TB, Digital Equipment Corporation, Maynard, Massachusetts; 1977.

4. OPRSTK, DEC-10 Notes, Computer Center, University of Pittsburgh, Pittsburgh, Pennsylvania; May, 1980.

5. PDP11/IAS RUNOFF, DECUS, Maynard, Massachusetts; August, 1977.

6. INTRODUCTION TO THE VIRTUAL MEMORY, Computer Center, University of Pittsburgh, Pittsburgh, Pennsylvania; March, 1977.

7. The HELP Files: SYS;PIP.HLP, SYS:SORT.HLP, SYS:OPRSTK.HLP, SYS:RUNOFF.HLP, SYS:RUNOFF.INS, Computer Center, University of Pittsburgh, Pittsburgh, Pennsylvania; 1980.

8. INTRODUCTION TO DEC SYSTEM-10: TIME-SHARING AND BATCH, T. W. Sze, University of Pittsburgh, Pittsburgh, Pennsylvania; First Edition, September, 1974, Second Edition, September, 1977.

CHAPTER 8

OPERATING SYSTEM COMMANDS

8.1 Introduction

The software system of the DEC System-10 contains language processors and a variety of service programs. The most important one is the operating system, also called the executive system, the supervisor or the monitor. It is a master program which exercises an overall control on the entire System. It performs the scheduling of users from the queue, supplies them with proper language processors and other system resources when requested, keeps account of charges, and performs many other service functions. An operating system command may be issued only when the user is in the monitor mode, which is indicated by the appearance of a prompt symbol "." (a period) on the terminal output. If the user is not yet in the monitor mode, he can get there simply by pressing the CTRL-C (^C) key on the terminal.

Before going into some details of the operating system commands, it is necessary to get acquainted with some terminology.

(1) Job The entire sequence of steps, beginning from the signing-on step and ending with the signing-off step, is called a job. Each job, while active, is assigned by the System with a job number, such as "Job 16". Within each job, the user can perform many functions, such as calling on system resources like tapes and disks, preparing and running programs, and communication with the System or other users.

(2) System device name Each system peripheral device has two names, its physical device name and its logical device name. They are explained below:

A. Physical device name Each input or output peripheral device associated with the System has a standard physical device name so that it can be referred to consistently. The format of a physical device name is:

DEVnnn:

where: DEV = three-character abbreviation assigned for a class of devices, for example, LPT for all line printers in the system.

nnn = zero to three-digit number indicating the numerical designation for a particular unit in a class of devices, such as "DTA010:" for DECtape drive Number 010.

: = a colon, an integral and terminating part of the device name.

If there is only one device in a particular class, the part "nnn" may be omitted. For example, "PLT:" is used to represent the system plotter, "PRG:" the Program Library, and "SYS:" the System Library. It may also be omitted in a multi-unit device name if only one such unit is assigned and available for general usage, such as "DSK:". A list of physical names of selected system devices is shown in Table 8.1.

Device	Physical Names
Array processor unit	APU:
Card Punch	CDP:
Card Reader	CDR:
DECtape Drive	DTA: DTA010:, DTA011:, etc.
Disk	DSK:
Line Printer	LPT: LPTS3:, LPTS6:, LPTS10:, etc.
Magtape Drives	MT7: (7-track drive) MT8: (9-track drive, 800/1600 BPI) MT9: (9-track drive, 1600/6250 BPI) MTA010:, MTA011:, etc.
Operator's Terminal	CPR: or TTY0:
Plotter	PLT:, PLT010:
System Library	SYS:
Program Library	PRG:
Engineering Library	ENG:
Terminals	TTY: TTY0:, TTY16:, TTY63:, etc.

Table 8.1 System Devices and Their Physical Names

B. Logical device name The user may also define the device with a name of his choice, which may or may not be the same as the device's standard physical name. Such a user-chosen name is called the logical device name. A logical device name may be assigned by either the ASSIGN command or the MOUNT command, as described in this chapter. Once a logical name is assigned, a device may be referred to by either its physical or logical name. Since the logical name may be chosen arbitrarily, the name chosen may already exist as a physical name of a different device. The conflict is resolved by the System that gives the logical name assignments precedence. Sometimes, it is beneficial to purposely cause such a conflict. For example, suppose a programming project has been completed after a great deal of effort on its preparation, debugging, compiling and documentation writeup. Suppose the program is designed to produce

its output on a line printer. Suppose now you wish to run this program, but you want a disk file output rather than a printer output. Instead of making extensive changes on the program and re-compiling it, you can simply call the disk a "printer" by giving the disk a logical name of the printer. Then, when you run the program, the System would take the disk as the "printer" and produce the output there.

The format of a logical device name is:

LOGDEV:

where LOGDEV is an one to six-character alphanumeric string, and the colon is an integral and terminating part of the name.

(3) Switch In most operating system commands, options or variations within each command are available. For example, to print a file on a line printer, a user can have a choice of printing it in single-space, double-space or triple-space, or specifying the number of copies he wants. These options are built into the operating system commands as option switches or simply switches. There are two general formats of switches:

/KEYWORD
/KEYWORD:argument

In the first format, the option does not require any other information, such as /FAST, /FORTRAN, etc. In the second format, the option will require a parameter specification, such as /SPACING:DOUBLE or /SINCE:22-JUL-1980 or /PROTECTION:155. Generally, these switches may be placed anywhere at the command keyword, and the order of the multiple switches is optional. However, when there is a list given in an Input/Output command, the placement of a switch in the command structure will make it either a "global switch" or a "local switch". The effect of a local switch only applies to the file it specifies. The effect of a global switch will extend to the rest of the command structure, unless its effect is overriden temporarily by a local switch or permanently by another global switch. When the following command is applied:

.PRINT PRG1.FOR,/COPIES:2 PRG2.FOR, PRG3.FOR, PRG4.FOR/COPIES:1, PRG5.FOR

1 copy each of PRG1.FOR and PRG4.FOR and 2 copies each of PRG2.FOR, PRG3.FOR and PRG5.FOR will be printed.

(4) DEFAULT CONDITION When an operating system command is executed, it runs a particular system program. Early in that program, if options are allowed, a set of initial conditions is established for these options. These initial conditions will remain unless they are replaced by the specified switches in the user-issued command. Thus, if user's option is the same as the initial conditions, there is no need for him to include such a switch in the command. Or, if the user is not familiar with the available options of a particular command, the command will be executed according to the established initial condition set. These conditions established by the System are called the default conditions of the switches. They are judiciously chosen to represent what an average user would want in a typical case. For example, in listing a file on the printer, single-spacing will be assumed if spacing option is not specified. Many default conditions are locally defined by a process called operating system generation, at which time the conditions are fixed according to such considerations as the installed system capacity, institutional operating policy, all aspects of the user population, and many local circumstances. Therefore, these default conditions may vary from one installation to another. Even at the same installation, they may vary from one time to another.

(5) File specification For the purpose of identifying a file, each file is given a name. Once the names are established, the system will maintain a directory so that users need not be concerned with the exact location on the disk for their files. For the DEC System-10, the format of a complete file specification is:

DEV: NAME.EXT [m,n] <xyz>

where: DEV: = name of device on which the file is stored. If this part is omitted in the complete specification, DSK: (the disk assigned to users) is assumed.

NAME = filename consisting of one to six alphanumeric characters with no embedded blanks.

.EXT = file extension consisting of zero (0) to three alphanumeric characters with no embedded blanks. If it contains zero characters, it is called a null extension. The period is an integral part of the extension.

[m,n] = the PPN of the person who created and owns the file. The default PPN is the current job's PPN. Note the use of square brackets.

<xyz> = a three-digfit protection code. Note the use of angular brackets.

The file extension is a part of file identification, used to indicate the type or language of the file. Although any zero to three-character combination can be used as a file extension for any file, the following are some most frequently used file extensions, and their meanings are recognized by the System.

Examples:

NEWTON.PIL	A PIL program file named NEWTON.
NEWTON.FOR	A FORTRAN program file named NEWTON.
NEWTON.REL	An object program compiled from NEWTON.FOR
FOR01.DAT	A data file named FOR01.

Symbols "*" and "?" are used as "wild cards" to represent a class of file names or extensions, as illustrated by the following examples:

Examples:

NEWTON.*	All files named NEWTON of any extension.
*.FOR	All FORTRAN files.
.	All files.
F????.DAT	All data files whose names are 5 characters or less and begin with F.

Dl2??.D??	All files whose names begin with "Dl2" and contain 5 characters or less, and whose extensions begin with the letter D and contain 3 or less characters.
Dl2??.*	All files whose names begin with "Dl2" and contain 5 characters or less.

The protection code is a 3-digit octal number xyz, each digit ranging from 0 to 7. Each digit defines a protection level of the file against a certain class of users:

x = protection level against the file owner himself.

y = protection level against users sharing the same project number.

z = protection level against the general public.

The level of protection ranges from level 0 to 7, and level-7 is the highest. The exact definition of each protection level is given below:

Code Digit	Access Protection*
7	No access privileges
6	Execute only
5	Level 6 + Read privilege
4	Level 5 + append privilege
3	Level 4 + update privilege
2	Level 3 + write privilege
1	Level 2 + rename privilege
0	Level 1 + change protection privilege

Access protection can be changed by executing RENAME or PROTECT monitor command (see Chapter 6) or the PIP program (see Chapter 7). Since there are 8 levels of protection in each of three classes of users, there are 512 different shades of protection-level combinations possible. Normally, one need only be concerned with a few commonly used codes:

Protection Codes	Applications
077,177	Strictly private and non-sharable, such as grade files maintained by an instructor.
057,177	Sharable within a project, for example, a program to be shared by all students in a course.
055,155	Sharable with the computer community, but the file may not be modified by anyone except the file owner.

The System assigns a default protection level of 057, set automatically by the computer if the person does not specify any protection code when he

---

*Subject to minor local variations. For example, at the University of Pittsburgh, access protection designated by the x-digit has been modified slightly.

creates the file. In some course work, instructors may arrange to have default protection level automatically set at 077. In such a case, the protection code of a student's file is 077 to his classmates, but is 057 to his instructor.

The Operating System of a computer is the most important and extensive software system. For DEC System-10, its Operating System contains more than a hundred commands, and some of its commands contain more than two dozen switches in each command. Studying and mastering the full set of commands can be an overwhelming task.

In the sections that follow in this chapter, a judiciously selected subset of these commands and a selected subset of their respective switches will be included. Since not every operating system command will be useful or meaningful to an average user, nor need he know every switch or available option, these subsets are chosen on the basis of what the author believes to be the most important and frequently used ones. The readers are referred to Reference 1 for a complete description of the operating system commands and their respective switches.

The discussions of the operating system commands will be functionally divided into six groups:

(1) Job initiation and termination commands

(2) Communication and status reporting commands

(3) Source file preparation commands

(4) Allocation of facilities commands

(5) Program execution and control commands

(6) File management commands

They are presented in the following sections.

JOB INITIATION AND TERMINATION

8.2 Job Initiation at a Remote Terminal

The sign-on procedure to initiate a job has been discussed in Section 1.8. For the purpose of completeness, they are again included here.

Once a user has a valid pair of ID numbers (the PPN) and has a valid password, he may now sign on at any remote terminal by following the procedure outlined below:

Hard-Wired Units	Dial-Up Units
(1) Turn on switches. Press C if there is no prompt symbol ".". After the prompt "." appears, type "I" (for INITIATE) and the following lines will be typed out on the terminal:	(1) Turn on switches and dial the computer number.* If the line is busy, there is a usual busy signal. When the call gets through, a high-pitch tone can be heard. Place the phone set on the seat of the acoustic coupler. Wait until the READY or CARRIER light comes on, type ↑C, and the following two lines will be typed out on the terminal:

```
PITT DEC-1099/A 63A.41B 15:36:41 TTY43 system 1237/1240
PLEASE LOGIN OR ATTACH
```

where "1099/A" indicates System A, "63A.41B" the monitor version, "15:36:41" the time of the day in 24-hour clock, "TTY43" the line number assigned. If "1099/B" appears instead of "1099/A", it means the user is in touch with System B. If the user finds himself in a wrong system, he requests a change by typing:

```
TTY SYSTEM B        or        TTY SYSTEM A
```

after the prompt symbol.

(2) Type the monitor command after the prompt symbol:

```
LOGIN m,n
```

or

```
LOGIN m/n
```

where m = project number, n = programmer number.

The difference between "m,n" and "m/n" in the two monitor commands is that the latter form will suppress the message of the day from the Computer Center when the sign-on procedure is completed. It is possible that you have seen the message several times already, and may not care to read it another time.

(3) Enter the password when requested. The password will be entered in a non-print mode, and the typed password will not appear on the terminal. This is to maintain the security of the password.

---

*For University of Pittsburgh users, dial (412) 621-5954.

If the entered password is an incorrect or invalid one, the system will respond with an error message and a request for the PPN. After supplying the PPN again, another password request will be made by the computer. The user has five chances to sign on correctly. After that number of unsuccessful trials, the job is killed, and the user must restart the entire procedure to sign on.

If the password is found to be valid, the system will respond with information on the status of the project, the last sign-on time and date, the time of day, and the "message of the day" from the Computer Center. The last item may be suppressed if the user uses the LOGIN command with the m/n specification.

After all preliminary reports are finished, a prompt symbol "." is printed on a new line, and the computer pauses and waits for input. The user is now connected to the computer at the monitor level, and the sign-on procedure is completed.

The following two cases are examples of sign-on. Explanatory remarks are also given along with the remote terminal printout. As used throughout this book, those lines entered by the users will be in *italics*:

Printout on Terminal	Remarks
. I	To initiate
PITT DEC-1099/A 63A.41B 16:19:17 TTY43 system 1237/1240	Computer response
.TTY SYSTEM B	Ask for System B
PITT DEC-1099/B 63A.41B 16:19:50 TTY43 system 1237/1240	System B response
.LOGIN 115103,320571	Sign-on command
JOB 35 PITT DEC-1099/B 63A.431B TTY43 Wed 7-May-80 1619	
Password: (*password*)	Enter password
Last login: 7-May-80 1617	Password valid
Usage ratio: 22.13 Units used: 33.5	
SYS B DOWN 0000-0800 MON MAY 12 FOR REGULAR HARDWARE MAINTENANCE	
SYS B DOWN 0000-0300 TUE MAY 13 FOR REGULAR SOFTWARE MAINTENANCE	
DUE TO HARDWARE PROBLEMS THE ARRAY PROCESSOR WILL BE TEMPORARILY OFF LINE UNTIL FURTHER NOTICE	Message of the day
.LOGIN 115103/320571	Sign-on command
JOB 23 PITT DEC-1099/B 63A.41B TTY43 Wed 7-May-80 1815	
Password: (your *password*)	Supply a password
Last login: 7-May-80 1619	
Usage ratio: 2.13 Units used: 33.5	On line

8.3 Password

To sign on the DEC-10 system, the required identifications are a valid PPN and the associated password. Security of PPNs is impossible because they are publicly displayed in many places - in LOGIN printout, in the file directory, in printout identification, etc. Thus the only real safeguard and security of a computer account is the password.

The need for protection against unauthorized use of your account by another person goes beyond accounting reasons. There have been numerous incidents of computer vandalism in the past. The most frequent vandalism was change or erasure of programs or data without the owner's knowledge.

The only protection against such unauthorized use is to install a password, to keep its security, and to change it frequently. As a matter of prudence and necessity, the user should change his password regularly as a standard practice and whenever he suspects the password is no longer secure.

Changing a password at a terminal can only be done at the LOGIN time by using either of the following LOGIN format:

```
LOGIN m,n/PASSWORD
```

or,

```
LOGIN m/n/PASSWORD
```

where "m" and "n" are the PPN. The following shows a sign-on session with a password change. Since the process is interactive, the explanation should be self-evident:

```
.LOGIN 115103/320571/PASSWORD
JOB 16 PITT DEC-1099/B 63A.41B TTY43 Wed 9-May-80 2003
Password:  Your old password
New Password:  Your new password

Retype for verification

New Password:  Your new password again
Last password update:  24-Apr-80 1255
Last login:  22-Apr-80 1642
Usage ratio:  0.84 Units used:  33.1

.
```

8.4 Job Termination at a Terminal

To leave the system, the user must terminate his job by supplying a monitor command KJOB ("to kill the job"). The system will respond by requesting a code letter for confirmation and file disposition. Thus, the command format for signing-off is:

```
.KJOB
CONFIRM:  Code Letter
```

A shortened form of this command is:

.K/*Code letter*

The most commonly used code letters in the KJOB command are:

F = fast signoff; save all files

D = fast signoff; delete all files. Computer will respond with A confirming question: "DELETE ALL FILES?" Answer YES and return the carriage.

P = preserve all files except temporary files.

H = HELP! Computer will respond with detailed instructions.

I = list file names, one at a time, and apply code letter decision individually. The code letters for individual decision are:

P = preserve the file

S = save the file

K = delete the file

Q = learn if over logout quota on this file

E = skip to next file and save this file if below logout quota for this file. If not below logout quota, a message is typed and the same file name is repeated.

H = HELP. Computer will respond with the above information on code letters.

While files are disposed per user's code letter instruction, the computer will make a check on logout quota, gather all usage and accounting information, terminate the user's job and print out a summary of the job. For example:

```
.K/F
JOB 16 [115103,320571] off TTY43  at 2032 9-May-80  Connect=29 Min
Disk R+W=83+76  Tape IO=0  Saved all files (450 blocks)
CPU 0:04  Core HWM=11P  Units=0.1263 ($9.48)
```

The printout indicates that this user, with PPN of 115103,320571, was assigned line 43 and job 16, signed off at 2032 on May 9, 1980. His terminal was connected to the system for 29 minutes, used CPU or computer time for 4 seconds. He used disk, but not magnetic tapes. He has 450 blocks of saved files. For this job, the highest core area used (HWM=High-Water-Mark) was 11 pages or 5.5K words, and the charge is 0.1263 unit or $9.48.

The "unit" is an accounting device which combines all charges of the service, including CPU time, disk usage, the length of connect time, the size of core used, and time of the day, and a base charge, each with an appropriate weighting factor to form an accounting formula.

COMMUNICATION AND STATUS REPORTING

8.5 Communication in the Time-Sharing System

Communication with the Computer Center staff and other users in the System is provided by several commands:

(1) SEND Command The SEND command enables a user to send a message from his terminal to another, including the system operator's terminal. The command format is as follows:

```
SEND dev:   message
SEND job n  message
```

or,

where dev: = TTYM:, the physical name of the destination terminal and M is its line number, e.g., TTY43:, TTY66:, etc.

or, = OPR: (for operator, same as TTY0:)

and n = job number at the destination

Example: Suppose at a remote station TTY20:, a message is sent to station TTY40:

```
.SEND TTY40: MEET YOU FOR LUNCH IN TEN MINUTES?
```

At the destination TTY40, the message will interrupt the job and is printed out:

```
;;TTY20:- MEET YOU FOR LUNCH IN TEN MINUTES?
```

The user can determine his own terminal line number by examining the LOGIN printout message or by issuing a command "PJOB." The command SYSTEM/J will print out the system active job status at the moment, including job numbers and line numbers.

(2) R MEMO Command This is a command for the user to communicate with the Computer Center staff for questions, suggestions, and complaints. When he completes his message, he terminates it by pressing the CTRL-Z (the CTRL and Z keys together) keys, and the user's terminal is returned to the monitor mode. If the message is long and pre-stored as a disk file, a user can load his stored message file by using an indirect file specification when message is requested in the program. An indirect file specification has a prefix of "@" before the standard file specification.

Example:

```
.R MEMO

Your [p,pn], TTY number, and the current date and time are automatically
recorded with your message.  A written response will be mailed to you
within one week.  Please provide a campus address, if possible.

Your Name:  T. W. SZE
Phone:  5418
Address:  339 BEH

Please type your message or indirect file spec. End it with a control Z.
Message goes in here....
^Z
EXIT
```

(3) To Send or Receive "Mail" The SEND command transmits message from one active terminal to another. Message can be sent to another user, whether or not he is currently on-line, by a "post office" system developed at the University of Pittsburgh.

In order to join this "post" system, a user must make some initial preparations. When he first joins the system, he should issue a monitor command of "R MAIL" which will respond, naturally, with a message of "NO MAIL POSTED." It will then go into the mail-sending sequence, which can be shorted out by a CTRL-Z or CTRL-C key. However, in doing so, a file is automatically prepared in the user's disk area with a file name of MAIL.BAG<144>. This is the user's mail box, without which he can send but cannot receive mail.

In this post system, the address of the receiver is his PPN. If the mail is directed to the Computer Center staff, the address may be one of the following, depending on the nature of the communication:

ACCOUNTING	CSMP	MACRO	REPAIR	SPSS
ALGOL	CTLYST	MAIL	RUNOFF	SSCRI
BASIC	FORTRAN	OPERATOR	SIMULA	SYSTEM
BATCH	F10	PIL	SITGO	TAPE
COPY	HELPER	PPN	SNOBOL	TAPELIBRARIAN
CREF	LINK	PROG	SOLO	TECHNICIAN
COBOL	LISP	PROGLIBRARIAN	SORT	TECO
COMPIL	LOADER	PROG L	SOS	UPDATE

To send a message through the post system, use the monitor command:

```
R POST
```

The procedure of sending mail is illustrated by the example below: The text in italics is typed by the user:

```
.R POST

TO:     114713,320571
SUBJ:   HOMEWORK DUE DATE

TYPE IN MESSAGE.  END WITH ^Z
WHEN IT ASSIGNMENT NO. 5 DUE?  ↑Z
TO:  ↑Z
```

The CTRL-Z closes the "letter" which is then stored in the receiver's file MAIL.BAG along with the information of sender's PPN, name, time of the day and the date. When one letter is completed by the CTRL-Z signal, the system responds with another "TO:" for the next letter, and the above process may be repeated for another letter. If there is no further letter to post, the user returns to the monitor mode by again pressing the CTRL-Z key.

To read these letters, issue a monitor command:

R MAIL

All new messages since the previous reading of mail will then be printed on the user's terminal. At the completion of the printout, the messages just read are emptied into another file MAIL.OLD and a new blank file MAIL.BAG is created to accept future mail. In the meantime, the system switches to the beginning of the POST sequence by typing out "TO:". The user at this point may send message out should he wish to answer his mail at that time.

The following is an illustration of how the person in the previous example might read and answer his mail:

```
.R MAIL

[122345,765432]  *DOE          14:28. JULY 27, 1980

SUBJ:  HOMEWORK DUE DATE

WHEN IS ASSIGNMENT NO. 5 DUE?

TO:    122345,765432
SUBJ:  HOMEWORK DUE DATE

TYPE IN MESSAGE.  END WITH ^Z
MY DEAR BOY, IT WAS DUE TWO WEEKS AGO.  ↑Z

TO:  ↑Z
```

If a user just wants to know if there is any new mail waiting since he checked it last time, he can issue this command:

```
R MAILX
```

The system will respond with a message of "MAIL WAITING ...", or just another prompt period, to signify whether or not there is mail waiting.

(4) The Computer Center also "posts" bulletins of general interest. These are stored in the physical device "SYS:" as news files. The files are updated frequently to announce the changes, bugs, new developments in some particular processor. As a result, new items may be added, and some old and non-newsworthy items may disappear. The user may check the directory of the news file by a monitor command of "DIRECTORY SYS:*.NWS" and the system will respond by listing a complete list of news files available in the device SYS:. A copy of the bulletin may be obtained on the user's terminal by issuing a monitor command "TYPE SYS:xxxxxx.NWS", or on the printer by issuing a command "PRINT SYS:xxxxxx.NWS", where "xxxxxx" is the name of the file chosen.

The System also maintains a set of files which contains helpful information on various programs and commands. They are called HELP-files and are generally quite voluminous. One can find out what HELP-files (with

extension of HLP) are available by a command of "DIRECTORY SYS:*.HLP", and a complete directory of HELP-files will be typed on the user's terminal. The user can then use "TYPE" or "PRINT" command to get a copy of the selected file.

8.6 Status Report Commands

The DEC System-10 keeps a wealth of data and records on its own operation and those of the users. some of these information may be useful to a user and can be made available by certain commands. These are listed below:

Command	Short Form	Explanation and Examples
PJOB	PJ	To print out user's job number. Example: .*PJ* 16
NJOB	N	To print out total number of active jobs on the system Example: .*N* 24
DAYTIME	DA	To type out the date and time in the format of: day-month-year hour:minute:second Example: .*DA* 16-JUL-80 16:56:59
TIME	TI	To type out the following items: 1. Total running time since the last TIME command. 2. Total running time and connect time of the job. 3. Total core usage in kilo-core-second Example: .*TI* 0.38 0.38 kilo-core-second=3, Minutes connected=3
CURRENT	C	To type user's current usage status Example: .*C* Usage ratio = 0.45 CPU allowance in this connect hour = :02:00 Current hour ends in 24 min CPU time remaining = :01:55
USESTAT	↑T	To type out six items of user usage information: 1. incremental day time in seconds 2. incremental run time in seconds 3. incremental read and write disk in blocks 4. name of program running 5. core size used 6. program counter address Example: .*↑T* DAY::13:43RUN:6.30RD:66WR:7 HELP 2P+1P^C SW PC:400672 Note: "incremental" means the differential since the last USESTAT command.

<table>
<tr><td>HELP</td><td>HE</td><td>There are three HELP formats:

1. HELP — Outputs the instruction for the receiving information.
2. HELP dev:* — Output both names of features that have available on-line documentation (HELP-files) and names of monitor commands. If dev: is DSK:, it can be omitted from the command format.
3. HELP NAME — Same as TYPE SYS:NAME.HLP</td></tr>
<tr><td>SYSTAT</td><td>SYS</td><td>The command format is: SYSTAT/switch

If the switch is not given, the entire system status will be typed out on the terminal. If switch is given and included in the command, a subset of system status report will be typed according to the following codes:

/Switch — Subset Information to Be Printed

/B — Busy device status
/J — Job status
/R — Remote station status
/S — Short job-status report
/X — Read the explanation of recent crash
/. — User's job status
/n — Status of job n
/#n — Status of TTYn
/[m,n] — Status of jobs submitted
/[m,*] — by specified PPNs.
/[*,n] — "*" is wild card.</td></tr>
<tr><td>R QUOLST</td><td></td><td>To print out the status of user's disk usage and quota. See Section 1.10 of Chapter 1 for details.</td></tr>
<tr><td>RESOURCES</td><td>RES</td><td>To print out the names of all available devices, except TTY's and PTY's, unless they are down,busy,non-existent.</td></tr>
<tr><td>WHERE</td><td>W</td><td>The complete format is: WHERE dev:
It outputs the station number at which the specified device is located.</td></tr>
</table>

SOURCE FILE PREPARATION

8.7 Source File Preparation Commands

Although there are many editing processors available on DEC System-10, only one editor is presented in this book. It is called the UPDATE (University of Pittsburgh DAta and Text Editor). The monitor command calling for this processor is:

```
UPDATE NAME.EXT
```

If NAME.EXT is a non-existent file, the command opens a new blank file on disk for creation, later to be named as NAME.EXT. If the file NAME.EXT already exists, this command opens that file for editing with the UPDATE editor. Commands and procedures of using the editor UPDATE are presented in Chapter 2.

There are other editors available on the System, such as the TECO. The monitor command to call for the TECO editor is "R TECO".

ALLOCATION OF FACILITIES

8.8 Facility Allocation by Monitor

The monitor allocates peripheral devices, file structure storage, and core memory to users on request and protects these allocated facilities from interference by other users. It maintains a pool of peripheral devices divided into two groups: unrestricted devices and restricted devices. Among the unrestricted devices are line printers, paper tape reader and punch, and disk. They are allocated on request at a when-available basis, and request for their allocation is really a reservation for their use. Actual usage of these devices is shared with other users on a queuing basis. On the other hand, restricted devices, such as magtape drives, will allow exclusive usage of the device when allocated. The exclusive usage continues until the device is returned to the pool. Therefore, a user must possess certain qualifications in order to be allocated with restricted devices. For example, he must have possession of a registered tape in order to use the tape drive.

Many of these peripheral devices are non-sharable at the same time. When a non-sharable device is assigned to a job, it is taken out of the monitor's pool and it will not be available to other users. Since non-sharable devices are scarce resources, the user should return them to the pool as soon as he completes his tasks with these devices. At any time, a user may find out what is available in the pool by a monitor command of:

```
RESOURCES
```

and the System will respond with the list of devices available by their physical device names. See example below:

```
.RESOURCES
D200,USRA,USRB,PLT010,MTA010,011,015,DTA010,011,012,013
```

Therefore, for all users in the general community, two types of commands are used for allocation of facilities: one for the unrestricted devices, and another for the restricted devices. In the latter, operator intervention is built into the command process to check on certain user qualifications. They are now discussed in some detail.

8.9 Allocation of Unrestricted Devices

An unrestricted device may be allocated upon a user's monitor command of:

```
ASSIGN  DEV:  LOGDEV:
```

where DEV: = physical device name, and
LOGDEV: = logical device name, optional in the format.

The physical and the logical names of devices have been defined and discussed in Section 8.1. Logical names of different devices must be different from each other, but a logical name may duplicate a physical name (not necessarily

representing the same device). In the latter case, the logical name will take precedence over a physical name. For example, when the following command is issued:

ASSIGN CDP: LPT:

the name "LPT:" now serves as both the logical name of the card punch and the physical name of the line printer. Since logical name takes precedence, system output will now be re-channeled into the card punch file preparation, even though the program executed after the ASSIGN command was originally designed for line printer output. Such an assignment command then becomes a convenient way of re-designating input/output devices of a program without having to modify the program itself. An alternate way is to intercept the output file before it is sent to the output device, rename it with an appropriate extension (e.g., CDP extension for card punch output), and then apply appropriate monitor command to produce the output. See more details on output command in Section 8.14.

Since the System already has assigned certain unrestricted devices as the standard input/output devices in the time-sharing mode, it is not necessary for a user to request their allocation. Therefore, the ASSIGN command is generally used for two purposes only: (1) to request a non-standard input/output device, or (2) to rename a device by a logical name which is referred to in an existing program.

A very useful variation to FORTRAN users is the ASSIGN command of the following format:

ASSIGN DEV: nn

where nn = device unit number in the READ/WRITE statements of a FORTRAN program. If a stored disk file is assigned this way, then DEV: is DSK:, and the filename must be FORnn.DAT, where "nn" ranges from 00 to a number depending on system installation. In FORTRAN-10, the upper range is 63.

Example: Suppose a FORTRAN program has been prepared in which the READ/WRITE statements are of the form: READ(5,f)list and WRITE(6,f)list. This program is now stored on the disk as SAMPLE.FOR. The following different cases show how a user can run the program and pre-select certain devices as input or output media:

	Remarks
.EXECUTE SAMPLE.FOR	Input/output devices will be standard devices, namely, the remote terminal of the user.
.ASSIGN LPT: 6 *.EXECUTE SAMPLE.FOR* *.PRINT *.LPT*	Obtain the output from the printer. Input is still via the user's terminal.
.ASSIGN DSK: 6 *.ASSIGN DSK: 5* *.EXECUTE SAMPLE.FOR*	During execution, input data will be from a stored file FOR05.DAT, and after the execution, output will be stored in a new file named FOR06.DAT.

In the batch mode, the selection of non-standard devices is much more limited. Aside from the standard devices, selection of unrestricted devices is essentially limited to the "DSK:", and the procedure outlined in Table 8.2 for

"DSK:" will also apply for the batch mode.

Device Name	Time-Sharing System Assignments	
	Unit 5	Unit 6
TTY:	Standard assignment	Standard assignment
CDR:	Procedure: 1. Prestore cards as a file, named as XXX.CDR, where XXX=1 to 3-character name. 2. Issue monitor commands: ASSIGN CDR: 5 SET CDR XXX 3. Execute FORTRAN program.	Not applicable
LPT:	Not applicable	Procedure: 1. Issue a monitor command: ASSIGN LPT: 6 2. Execute FORTRAN program. 3. Output will be stored as a disk file Q????.LPT, where "????" are four characters arbitrarily assigned by the System. 4. Use either of the following monitor commands to get the printer output: PRINT *.LPT QUEUE *.LPT
DSK:	Procedures: 1. Prepare ahead a data input file, which must be named as FOR05.DAT 2. Issue a command: ASSIGN DSK: 5 3. Execute Fortran program.	Procedures: 1. Issue a monitor command: ASSIGN DSK: 6 2. Execute FORTRAN program. The output will be stored on disk as FOR06.DAT. If there was a previous FOR06.DAT on disk, the new file will replace it without warning. BEWARE!

Table 8.2 Assignment of Unrestricted Devices for FORTRAN Program Execution in the Time-Sharing Mode

A device, once ASSIGNed, will remain assigned, until the user issues a command of DEASSIGN to release it. The format of the DEASSIGN is:

```
DEASSIGN  DEV:
```

or,

```
DEASSIGN
```

where DEV: =either the logical or physical device name of the specified device. If it is not specified, all devices assigned to the user's job, except the remote terminal, will be released.

8.10 Allocation of Restricted Devices

A restricted system device is one where an operator intervention during its usage is necessary in order to determine whether the requesting user is eligible for the device. There are two types of restricted devices that may be of interest to the readers of this book: the DECtape drives and the magtape drives. The latter includes both 7-track and 9-track drives.

In general, when a user uses tapes in his processing, he goes through the following steps:

Tape Processing Steps	Monitor Commands*
(1) Reserve the necessary number of tape drives.	DRIVES
(2) Ask the operator to mount physically a designated reel of tape on a reserved tape drive.	MOUNT
(3) After the completion of tape processing of the reel, ask the operator to remove the reel from the tape drive. User will still retain the usage of the tape drive at this point.	DISMOUNT
(4) If the user has further tape processing to do, he will repeat step 2 and step 3.	MOUNT & DISMOUNT cycles
(5) When the tape tasks are finished, the user will release the reservation of tape drives and return them to the system pool.	

For more details of tape processing and handling, the readers are referred to Chapter 10. We will now discuss the monitor commands required to perform these steps.

---

*These commands are enhancement of monitor of the standard DEC-10 software, and were developed by the staff of the Pitt Computer Center. DRIVES and the UNDRIVES commands are new, and MOUNT command contains additional access restriction enforcement not available in the original DEC version.

(1) DRIVES and UNDRIVES Commands

To request reservation of tape drives, a PITT-developed monitor command should be issued which has a format of:

```
DRIVES  DEV(n), DEV(n),...
```

where DEV = physical name (without colon) of a restricted device, which is any of the following:

DTA = DECtape drive
MT7 = 7-track magtape drive
MT8 = 9-track magtape drive, 800 or 1600 bpi only
MT9 = 9-track magtape drive, 1600 or 6250 bpi only

and (n) = number of drives requested. If n=1, "(n)" may be omitted from the command.

To return the devices to the system pool, the command UNDRIVES has a similar format:

```
UNDRIVES DEV(n), DEV(n), ...
```

To avoid accidental release of tape drives, the command UNDRIVES DEV(n) will be ignored if the said drives still have tapes mounted on them, and if no DISMOUNT commands have been issued yet.

The commands DRIVES and UNDRIVES can be issued without any argument, and they will have somewhat different meanings:

DRIVES and UNDRIVES Commands With No Argument	Function
DRIVES	To report the status of the user's current allocation of tape drives.
UNDRIVES	To release all tape drives regardless of whether there are tapes still mounted on them or not. Such a command will force a DISMOUNT action on all tapes, and return all drives to the system pool. This is also a standard procedure in the KJOB to allow a quick exit from the System.

The following points will also be helpful in using the DRIVES and UNDRIVES commands:

A. The DRIVES command is not accumulative. If two or more DRIVES DEV commands are given in succession, only the last one will be in force, because any DRIVES DEV command always cancels out the previous one. Therefore, the drives needed for one tape task should always be requested in one single DRIVES command, and not piecemeal in several commands. Note the difference between the two following cases:

Case 1:
```
.DRIVES DTA(2),MT9
MTA013,DTA010,DTA011 ALLOCATED
```

Case 2:
```
.DRIVES DTA(2)
DTA010,DTA011 ALLOCATED
.DRIVES MT9
MTA013 ALLOCATED
.DRIVES
MTA013 ALLOCATED
```

In case 2, the second DRIVES command cancels out the first one. At the end, only the second request by itself is honored.

B. The System will not make a partial allocation to a DRIVES request. If there is not a sufficient number of requested devices currently free in the pool, the System will respond with a message as shown below:

```
.DRIVES DTA, MT9(2)
DRIVES NOT AVAILABLE NOW, WAITING....(^C^C TO EXIT)
```

At this point, the user has two options: One is to wait. Then he is not able to do anything at the terminal. The other is to cancel the request by pressing the CTRL-C key twice or more, and the user can submit another request sometime later. Naturally, such options are not available to BATCH users because they will not have the opportunity of such interactions.

C. Unlike the DRIVES DEV command, the UNDRIVES DEV is accumulative. For example, suppose a user has acquired 4 DECtape drives by a previous DRIVES command. If he issues a command UNDRIVES DTA, he will be left with 3 drives if the release request is successful. If he then issues another command UNDRIVES DTA(2), he will be left with just one drive.

D. Whether a release command UNDRIVE DEV will be successfully executed depends on whether there are still tapes mounted on the referred drives. Let us denote those devices being "idle" if there are no tapes mounted on them at the time. The result of the command UNDRIVES DEV(n) depends on the number of idle devices at the time, because with that command only the idle devices are released.

a. If n = number of idle specified device, all such devices are released.

b. if n<number of idle specified device, the UNDRIVES command will arbitrarily release n idle devices, but the user will not know which ones have been released unless he issues a new DRIVES command with no argument to inquire about the new allocation status.

c. If n>number of idle specified device, the UNDRIVES command will release all idle devices. Again no message is returned, and the user must use the DRIVES command to find out about the new allocation status.

(2) MOUNT and DISMOUNT Commands

By a MOUNT command, a user requests the operator to mount a tape at a designated tape drive. It has a form of:

```
MOUNT DEV:LOGDEV/switches
```

where DEV: = the physical name of the tape drive, and
LOGDEV = the logical name assigned by the user

The following are several more frequently used switches:

Switches	Explanations
/VID:Xnnnnnn	Visual identification. At the University of Pittsburgh, tape registry numbers are used as visual ID. They are decimal numbers (six digits maximum) with A-prefix for DECtapes and B-prefix for magtapes, for examples: A1234 and B313. This is the only way a user can specify which tape he wants.
/WENABLE	For "Write-enable". The tape will be available for both read and write operations. Its short form is "/WE".
/WLOCK	For "Write-lock". The tape will be for read-purpose only. Its short form is "WL".

Mounting of tape is a manual procedure, and the operator has only the visual ID to tell whether he has the right tape. Since human errors do occur, there is always a chance of an operator mounting a wrong tape, and the subsequent read-write operation will cause irreversible damages to the information storage. Each computer installation generally designs additional security measures to reduce the chance. At the University of Pittsburgh, additional security of tape access is implemented through a standardized tape registry, tape labeling and a modification of the MOUNT command.

The visual ID of each tape is standardized as a tape registry number, in the form of Xnnnnnn, where "X" is either "A" (for DECtapes) or "B" (for magtapes), and "nnnnnn" is a decimal number of maximum six-digits. During a "labeling" process, the numerical part of registry is recorded on the first file of a DECtape or a 9-track magtape. In executing the MOUNT command (PITT modified version), not only the operator will search for the right tape by the VID identification, but also the first file will be read by the System. Thus the number read from the first file may be compared with a VID given by the user. If the two numbers do not agree, the tape job is aborted. Details of how a tape may be "labeled" are given in Chapter 10.

With these modifications, the MOUNT command used at the PITT facility has the following additional switches:

/SL Standard label. This switch will instruct the System to check the label against the VID given. Actually this is a standard operation, even if no switch is specified. In other words, this is the default switch.

/NL No label. This switch informs the System that the tape is not labeled, for example, as the tape is brought from another institution.

Tape users should be aware that this security system is not applicable to 7-track magtape because of difference in recording techniques. When a switch relating to label is given in using a 7-track tape drive, the label switch will simply be ignored.

To dismount a tape from the drive, a user may issue a command of:

```
DISMOUNT  dev:
```

where dev: = previously MOUNTed device name, either physical or logical. After a DISMOUNT command is issued, the tape mounted will be removed by the operator and returned to its storage. The user, however, retains the use of the tape drive, and he may mount another tape for further processing.

Example: The following shows a typical case of tape processing:

```
                                    Comments
.RESOURCES                          To check on availability of devices
D200,PLT010,MTA010,013,015,DTA010,011,012,013

.DRIVES DTA(2)                      To request2 DTAs.  DTA010 and DTA011 are
DTA010,DTA011 ALLOCATED             available and assigned.

.MOUNT DTA:T1/WE/VID:A1004          To mount tape  A1004 on one tape drive, and
Request queued                      name it as T1.
Waiting... ^C^C to exit             Either wait or issue 2 ^C to get back
                                    to the monitor


.MOUNT DTA:t2/WE/VID:A1005          To  mount another tape, and name it as T2.
Request queued
Waiting... ^C^C to exit
↑C
↑C

[MNT - DTA010 (T1) mounted]         Mounting T1 message
[MNT - DTA011 (T2) mounted]         Mounting T2 message

     (Tape Processing
        done here)

.DISMOUNT T1                        Dismount T1. Colon is optional.
T1 Dismounted                       Returned message.

.DISMOUNT T2                        Dismount T2.
T2 Dismounted

  (More MOUNT and DISMOUNT sequence)

.UNDRIVES                           To return the DTAs to the System.
```

8.11 Remote Terminal Control Commands

A remote terminal is the most important peripheral device to a time-sharing user. Its principal characteristics were described in Chapter 1. When a terminal is connected to the System, a number of its operating conditions are initialized automatically, such as the right margin, the tab positions, etc. These conditions, however, may not always work well with certain terminals, because there are a wide varieties of terminals the System can support. Even on the same terminal, one user's requirements may not be the same as those of another.

The SET TTY (short form TTY) command allows a user to declare properties of his terminal, and it has the form of:

TTY *keyword*	or	SET TTY *keyword*

where keyword = either of the complementary pair of keywords for TTY properties. Table 8.3 shows a list of keywords in the TTY command.

Example: *.TTY WIDTH 132*
Function: Set the right margin of the remote terminal at column 132.

Example: *.TTY FILL 2*
Function: Some terminals suffer from timing problems at 300 bauds speed. The symptons are missing characters or overprints at the beginning of each line. The remedy is to delay the transmission of information after each long carriage or paper movement, such as carriage return or form feed, by inserting dummy or "filler" non-print characters. The fillers do nothing except to take up time. The idea is to allow the print head enough time to get into position when the transmission of information is resumed. The number of filler characters falls into four classes: Class 0, 1, 2, and 3 with the higher classes having more filler characters. Reference 1 gives a detailed table on the exact number of fillers in each class for every carriage or paper movement action. Since inserting fillers will slow down the print throughput, the filler class should be chosen just high enough to overcome the timing problem, if one exists. This can be easily determined by experimenting with each class.

TTY Keyword	System Default	Explanations
ALTMODE	X	Converts ALTMODE codes of octal 175 & 176 to ASCII code of 033.
NO ALTMODE		No conversion
BLANKS	X	Allows blank line printout.
NO BLANKS		Suppress blank lines. Useful in CRT terminal to increase output that fits on the screen.
CRLF	X	When a line reaches the right margin, the carriage will automatically return and advance one line.
NO CRLF		Suppress carriage return even when the right margin has been reached.
ECHO	X	Terminal will echo print the input characters.
NO ECHO		Suppress the echo print.
FILL n	n=0	Insert filler characters after each carriage-return or tabbing operation to correct timing problems. Filler insertion is of class n. See more explanation in the example below.
NO FILL	X	Same as n=0
GAG		Message transmitted by SEND command cannot be received at this terminal.
NO GAG	X	Opposite of GAG.
LC		Opposite of UC
NO LC	X	Transmits all lower cases as upper cases.
PAGE		After this command is issued, the user will have the ability to temporarily suspend system typeout without losing it. The key CTRL-S suspends the typeout, and CTRL-Q restores it.
NO PAGE	X	Disables the CTRL-S and CTRL-Q keys.
TAB	X	System sets up standard tab settings. Actual settings vary with each installation.
NO TAB		The monitor simulates TAB output from program by sending the necessary number of SPACES.
UC	X	Same as NO LC
NO UC		Same as LC
WIDTH n	72	The carriage width (the point at which a free carriage return is inserted) is set to n. "n" ranges from 17 to 200.
SYSTEM A		Applied before LOGIN command to select one of the
SYSTEM B		two systems.

Table 8.3 TTY Command Keywords

Example: *.TTY GAG*
Function: After this command is given, the user's terminal cannot print out messages sent by other users' SEND commands. This is useful when the user has an important output to prepare and he does not want any printed messages to spoil his output.

Example: *.TTY PAGE*
Function: This is particularly useful in a CRT terminal application to allow temporary suspension of output so that the user can read it before it rolls off and disappears from the screen. It is also useful in conventional terminal to create a pause during the typeout to allow the user to make some manual adjustment of the terminal, for example, to advance the paper to the next page.

Example:
```
.TTY PAGE
.TTY GAG
.TTY WIDTH 132
```
Function: Unless one keyword cancels out the effect of another previously given keyword, the TTY commands have an accumulative effect.

Example: *.TTY LC*
Function: For terminals having lower-case capabilities, this command will set the terminal at the lower-case mode, and the terminal acts like a conventional typewriter. The upper case character may be generated only if the shift key is depressed at the same time.

8.12 Execution and Related Commands

To execute a source program stored on disk, such as a program written in FORTRAN, the program is first compiled and an object program is generated and stored on disk. This object program, called a relocatable binary file, or a REL file, is then loaded into the user's core along with any subprograms of the System called by the program. Execution will then begin at an address of the core determined by the compiler and the loader. Therefore, the execution process goes through three stages: the compiling, the loading, and then the execution. Similarly, the loading goes through two stages: the compiling and the loading. Of course, the user may also request just the compiling to be done. The monitor commands to perform these functions are listed next. The "list" in the command format may be either a single file specification or a list of files separated by commas.

Command Format	Explanations
COMPILE list	To compile the source program(s) and store the REL file(s) on disk. No execution.
LOAD list	To compile if necessary, and then to load the REL files and the needed System subprograms and user-supplied subprograms in user's core. No execution.
EXECUTE list	To compile the source program(s) if necessary; store the REL file(s); load them along with all needed subprograms into the core; then execute.
START	To begin execution after a REL file has been LOADed. A LOAD command followed by a START command is equivalent to an EXECUTE command.

When a file is created for storage on disk, the directory carries the information of its creation date and time. When a command COMPILE, LOAD or EXECUTE is issued, the System will first search in the user's disk area to see if there is a REL file bearing the same name. If there is such a REL file on disk and if its creation date and time is newer than that of the source program, the compiling is simply bypassed because the REL file is still valid. The purpose is to avoid unnecessary compiling which can be quite costly.

A number of command switches are available, and a selected subset is listed below. As seen in Table 8.4, these switches are common for the commands listed except the LIBRARY switch.

Switch	Command			Function
	COMP	LOAD	EXEC	
/COMPILE	X	X	X	To force a compiling of the file even if a REL file already exists with a newer date and time than that of the source file.
/CREF	X	X	X	To produce a cross-reference listing file on the disk for each compiled file for later processing by the CREF program.
/F10	X	X	X	To use the FORTRAN-10 compiler. This is a default switch at Pitt.
/F40	X	X	X	To use the F40 compiler.
/LIBRARY		X	X	To load the files in library search mode. See more explanations in the examples.
/DEBUG:BOUNDS	X	X	X	To report if subscripts of array are out of bounds as defined for the array in the DIMENSION statement.

Table 8.4 Selected Switches for the EXECUTE Command

Example: Suppose a REL file has been prepared that contains fifty subprograms needed in a course. However, at any single application, only a few are really needed. Suppose this package is now named as COMMON.REL. At each application, a user will prepare a main program, in which he calls certain subprograms from the package. In executing his program the user should issue either of these two commands:

```
.EXECUTE MAIN.FOR, COMMON.REL
```

or,

```
.EXECUTE MAIN.FOR, COMMON.REL/LIBRARY
```

With the former command, the entire COMMON.REL is loaded into the user's core--all fifty routines--even though only one or two may be needed by the main program. In the latter form, only those routines needed by the main program are loaded. When a library package is large, the use of the LIBRARY switch will spell the difference of whether there is enough core to run the user's job.

When a program has been thoroughly debugged, and if the program will be run many times or shared by many people, the following is a more efficient way of executing it.

The program and its subprograms will be compiled and loaded in the usual way by a LOAD command. After the LOAD operation, the "core image" may be saved as a file bearing an extension of EXE by issuing the command:

```
SAVE NAME
```

and the save file will have a name of *NAME.EXE*. Any subsequent execution of the program may be done by issuing the command:

```
RUN  NAME[m,n]
```

where [m,n] is the PPN of the file owner, and may be omitted if the user has the file in his own disk area.

The main advantage of executing a program this way is to eliminate all preliminaries of compiling and loading. Besides, only one file name after the RUN command need be specified.

Example: Observe the following sequence with comments given:

```
                                                    Comments

.LOAD MAIN.FOR, SUB1.FOR, SUB2.FOR, SUB3.FOR, SUB4.FOR, COMMON.REL/LIBRARY
(Compiling of each FORTRAN program takes place here.)
(Loading of all REL files takes place next.)

.SAVE MESS                                  Save the core image as MESS.EXE
Job Saved
.START                                      Execute the program
(Execution follows)
```

Subsequent execution of the same program:

```
.RUN MESS
```

The same idea is extended to running the System program by using the R command of the form:

```
R    MESS
```

where *NAME* is an EXE file of the System. Thus, the commands "R UPDATE", "R PIL", etc. are among the applications of the R-command.

Programs submitted for batch job execution may be submitted through the time-sharing terminal by a command developed at Pitt:

```
OPRSTK  NAME.EXT
```

where NAME.EXT is the name of the control file. See Chapter 9 for details.

FILE MANAGEMENT AND CONTROL

8.13 File Management Commands

In the general specification of a file:

DEV: NAME.EXT [m,n] <xyz>

all components in the form except "NAME" part may be omitted. When a certain part is omitted, it has the following default interpretation:

Omitting	Means:
dev:	The device is DSK:
.EXT	The file has a null extension.
[m,n]	The file belongs to the user.
<xyz>	A file is uniquely specified without a protection code designation. However, if a file is created without specifying a protection code, a default code of 057 is given to it.

In addition, the character "*" and "?" serve as the "wild cards" in the format of file specifications, as was discussed in Section 8.1.

Various file management commands will be discussed next.

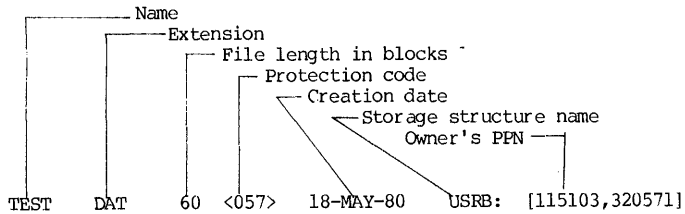
(1) DIRECT (DIR) Command This command will output a listing in file specifications, sizes in blocks, protection codes, structure names, creation dates, etc. The complete form of the command is:

DIRECT *OUTPUT = input list/switches*

where "OUTPUT=" (including the equal sign) is the output device and file name, and "input list" is a single file or a string of files. If the TTY is the output device, the part "OUTPUT=" may be omitted. If an output file name is given, the default device is DSK: If an output file is not given and one is needed, the default file name is HHMMSS.DIR where HHMMSS is the time of the day when the DIRECTORY command is given. Several examples are shown below:

.DIRECT	Print out a directory of all stored files.
*.DIRECT NAME.**	Directory of all files with the name NAME.
*.DIR *.EXT*	Directory of all files with the extension EXT.
.DIR FL1.EX1,FL2.EX2,...	Directory of individual files.

A typical printout of the directory is shown below:



The last line is a summary line of the files in the DIRECTORY command. Switches provide a user a wide selection of file categories and printout formats. Some of the switches are included below:

Switch	Argument	Directory Printed
/BEFORE: /SINCE:	date:time date:time	Files created during the specified time.
/DETAIL		To print out all details of a file lookup.
/FAST /NORMAL /SLOW		To print a short form of directory. To print a normal form. To print out a full listing.
/HELP		To print out all available switches.
/LIST		To list the directory on the line printer.
/SUMMARY		To print out just the summary line.
/WIDTH:	N	To output several entries on a single line to make output N-column wide. Default N is 64.
/WORDS		To print out the size of files in words instead of in blocks.

Example: *.DIRECT LIST.DIR = *.DIR*
Function: Store the directory of all FORTRAN files as LIST.DIR.

Example: *.DIRECT *.FOR/FAST/WIDTH*
Function: Print out the directory of all FORTRAN files on user's terminal with a format of names and extensions only, four entries per line.

Example: Suppose a DECtape has been mounted by a MOUNT command, and it has been named with a logical name of T1. The following command will list all FORTRAN files stored on that tape:
.DIRECT T1:.FOR*

(2) DELETE (DEL) Command This command will delete one or more files from the disk or the DECtape, and remove their entries from the directory. It has a similar format as that of the DIRECT command, namely:

```
DELETE list
```

where "list" = a single file or a group of files. As in the case of DIRECT command, the following are some of the most frequently used forms:

```
DELETE FL1.EX1, FL2.EX2,...
DELETE NAME.*
DELETE    *.EXT
DELETE    *.*
```

After the deletion of files is completed, computer will respond with a report of the file names, extensions, and total disk storage blocks recovered.

Example: *.DEL *.TMP, *.BAK*
Function: Delete all temporary and backup files created after a successful editing session using UPDATE editor, or after these files have served their usefulness.

(3) RENAME (REN) Command This command will change one or more items of the file specifications on the disk or DECtape. The items that may be changed by this command are filename, extension, protection code, or combinations thereof. The form of the command is:

```
RENAME new1=old1, new2=old2, ...
```

When this command is executed, the file specificaiton of "old1" is changed to "new1", "old2" to "new2", etc.

The old and the new file specifications must bear or imply a one-to-one correspondence, especially when there is a "wild card" representation. For example, the command ".RENAME NEW.FOR=*.FOR" would be an incorrect command.

Example: *.RENAME ISSAC.FOR = NEWTON.FOR*
Function: To rename a file from the name NEWTON.FOR to ISSAC.FOR, and keep the same protection code.

Example: *.RENAME *.FOR<157> = *.FOR*
Function: to give all FORTRAN files a protection code of 157. If a file already has a protection code of 157, the command is still executed despite being superfluous.

Example: *.REN T1:*.FOR = T1:*.F4*
Function: To rename all extensions of F4 to FOR for files stored on DECtape, previously MOUNTed and given a logical name of T1.

(4) PROTECT (PROT) Command This command will alter the protection codes of specified files. Its format is:

```
PROTECT  file1<xyz>, file2<xyz>, ...
```

where "xyz" is the new protection code assigned. This command is equivalent to "RENAME file1<xyz>=file1,fiel2<xyz>=file2,..." In fact they are executed by the same program.

(5) PRESERVE (PRE) Command This command will rename the file to change its protection code from <xyz> to either <157> or to <lyz> depending on the local installation practice. The purpose of "preserving" a file is to raise its protection level relative to others in the disk. In a batch run, if the disk storage is over the logout quota (See Section 1.10) at the end of a batch job, the System will kill off excess files ruthlessly according to an established order of priority as given in Section 8.4. However, preserving a file merely makes it less vulnerable--it does not make it untouchable. If all files are PRESERVEd in the disk, the idea of relative protection level is lost, and one file is just as vulnerable as another. In general, all source language files should be preserved, and all files that can be re-generated (by re-compiling or by re-running the source program) need not be preserved unless it will be costly to re-generate them. The command format of the PRESERVE command is:

```
PRESERVE  list
```

where "list" is a list of file specifications to be preserved.

(6) COPY (COP) Command This command will duplicate as a single file from one or more source components. The format of the command is as follows:

```
COPY  output file spec = input1, input2, ...
```

where the file specifications on both sides of the equal sign have the standard form. The equal sign is required in the format, as it separates the destination side from the source side. The files at the source may have wild card construction, and they must be accessible to the user if they are stored under other PPNs. The order of input files is important, because they will be merged into an output file in the precise order listed in the command.

The six file management monitor commands are among the most frequently used commands. Actually, all of these commands except DIRECT run a system service program PIP and utilize some of its salient features. The PIP program can do a great deal more than that presented here, and it is one of the most versatile and important service programs. Details of PIP were given in Chapter 7.

There are a number of points that may be of interest:

A. When a file of another PPN is included in the file management command, that PPN must be specified. If several files have the same PPN specification, command structure can be simplified by moving the [m,n] part in front of these files. Thus the following two commands are equivalent:

B. In a multi-user time-sharing and batch system, the security of stored files has increasingly become such a serious problem that legislations are being considered at the federal level. To copy a copyrighted file without permission can lead to civil court action. To discourage "snooping" and abuses, file management commands are often modified and curtailed when another user's PPN is specified in the command. For example, when a user has logined under his own PPN of [ml,nl], the following commands will be rejected by the System and considered as "snooping" abuses:

```
.DIRECT  [m2,n2]
.DIRECT  *.FOR[m2,n2]
.DIRECT  X???.REL[m2,n2]
.COPY    *.* - *.*[m2,n2]
.COPY    *.BAS = *.BAS[m2,n2]
etc.
```

C. While the examples of these commands have mainly concentrated on the management of disk files, they are equally applicable to DECtape file management. However, before these commands may be applied to DECtape files, the tapes must be mounted by the DRIVE and the MOUNT commands. The following shows a typical session of DECtape file management. Computer message printout is omitted to save space:

```
.DRIVES DTA
.MOUNT DTA:T1/WE/VID:A1004
.RENAME T1:NAME1.FOR = T1:NAME2.FOR
.COPY SAMPLE.DAT = DATA1.DAT, T1:DATA2.DAT
.DELETE T1:DATA2.DAT
.DISMOUNT T1
.UNDRIVE
```

8.14 File Output Commands

A file on disk or DECtape can be reproduced as an output via one of many output devices. These output monitor commands are now tabulated below:

File Output Monitor Command	Function
TYPE *list*	To produce a typed copy of files listed in tne *list* on the user's terminal.
PRINT *list*	To produce a printed copy of files listed in the *list* on the line printer.
CPUNCH *list*	To produce a punched card deck of files listed in the *list* on the system card punch.
TPUNCH *list*	To produce a roll of punched paper tape of the files listed in the *list* at the system paper tape punch unit.
PLOT *list*	To produce a plot of files listed in the *list* on the system Calcomp plotter.

where "list" is a single file specification or a series of file specifications. Except for the TYPE command which produces the output at the user's terminal,

all other commands listed above use the system output facilities. As the facilities are shared by all users, by necessity a traffic control scheme must be established for orderly execution. Priority rules must be set up so that the facilities will not be monopolized by large-output jobs. These considerations led to the idea of setting up a queuing line of output jobs, and the design of a strategy of assigning priorities to jobs in the queuing line. Furthermore, the need for establishing priority and queuing also exits for batch jobs. Therefore, one single monitor command may be used to request output, and the command will substitute for the above listed commands (all except TYPE). The command QUEUE will be presented next.

8.15 The QUEUE Command

The QUEUE command allows the user to make entries in several system queues - the input queue for the batch system* and the output spooling queues for the line printer, the card punch, the paper tape punch, and the plotter. The QUEUE command with appropriate switches also reports or modifies the status of the queue entries.

The general form of a QUEUE command is as follows:

```
QUEUE QLINE:JOBNAME = NAME.EXT/Switches
```

The QLINE parameter selects the queuing line. At the University of Pittsburgh, there are five queuing lines incorporated into the QUEUE command: the batch queue, the line printer queue, the card punch queue, the paper tape punch queue and the plotter queue. They are discussed in detail next.

*At the University of Pittsburgh, the capability of the QUEUE command to submit an input batch job is disabled. The entry of input queue in the batch system is now done by the OPRSTK command (see Chapters 7 and 9). However, the QUEUE command may still be used to inquire about or modify the status of batch jobs in the input queues.

First, the QLINE: parameters are outlined below:

QLINE:	Short Form	Explanations
INP:	I:	Batch input queue.
LPT:	L:	Line printer output queue. For printer jobs, station number can be specified to designate a particular printer in the form of LPTSn: This QLINE is optional for all print jobs, except when the file extension is any of: CDP, PTP, or PLT.
CDP:	C:	Card punch output queue. This QLINE is optional if the file extension is CDP.
PTP:	PT:	Paper tape punch output queue. This QLINE is optional if the file extension is PTP.
PLT:	PL:	Plotter output queue. This QLINE is optional if the file extension is PLT.

Other parameters in the command string are:

JOBNAME = name of the job being entered into the queue. This part is optional. If omitted, the jobname given by the System is the name of the first file in the request.

NAME.EXT = file specifications which may include [m,n] for other PPN. Wild card construction is also allowed.

/Switches = switches that control the action of the command.

The main advantage of using a QUEUE command is that it unifies and replaces many input/output monitor commands. If the rule on the file extension is observed (namely, "The extensions CDP, PTP, and PLT are exclusively reserved for output files of card punch, paper tape punch, and plotter, respectively"), the following substitution of commands may be made:

Monitor Command	May be Substituted By:
SUBMIT *list**	QUEUE *list*
PRINT *list*	QUEUE *list*
CPUNCH *list*	QUEUE *list*
TPUNCH *list*	QUEUE *list*
PLOT *list*	QUEUE *list*

Thus, if a file is named with an appropriate extension, the QUEUE command will automatically enter it in the proper output queue and the QLINE in the command structure may be omitted.

---

*"SUBMIT list" command for submitting batch jobs is disabled at Pitt.

There are three types of QUEUE switches:

(1) Queue operation swithces These switches define the queue operations. Only one of this type may be placed in a QUEUE command. It may appear anywhere in the command string.

(2) General queue switches These switches generally affect the scheduling of the queue entries. Each switch of this type may appear only once in a command string but it affects the entire request. It may appear anywhere in the command string.

(3) File-control switches These switches affect the individual files in a request and must be adjacent to the file specification in the command string. Different placements of these switches in the command string have different meanings. Normally, such a switch is placed immediately after a file specification, and that switch only controls that particular file. If such a switch is placed immediately before a group of file specifications, that switch will control all files that follow until restored or changed by another switch. See the following examples:

Examples	Remarks
F1.EXT,F2.EXT/COPIES:2,F3.EXT	Requesting 2 copies of F2, 1 copy of F1 and F3
/COPIES:2 F1.EXT,F2.EXT,F3.EXT	Requesting 2 copies of F1,F2,F3
/COPIES:2 F1.EXT, F2.EXT, F3.EXT/COPIES:1	Requesting 2 copies of F1 and F2, 1 copy of F3

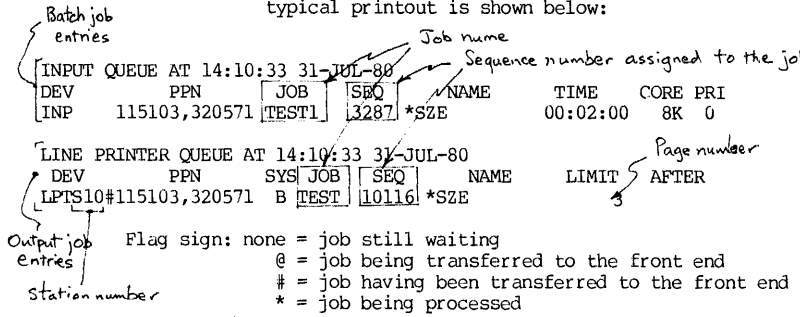
Actually, this is also true for a device name, a PPN, or a file extension.

The number of switches with their respective arguments is quite large. However, not every one will be useful or meaningful to an average user. Consistent with the purpose of this book, only a judiciously chosen subset of these switches will be presented, along with appropriate remarks, examples, and explanations. For complete information, the readers are referred to References 1, 2, and 6.

Note that if an argument of a switch is omitted, a default value is assumed only if the colon is also omitted. Otherwise, the value of the argument will be assumed by the System as zero, not the default value. For this reason, it may be more prudent to consider the colon to be a part of the argument, rather than a part of the switch. The selected switches are now given in three tables, categorized in the types mentioned above as Table 8.5, 8.6 and 8.7. The keyword for each switch may be shortened. The underscored part of the switch is the minimum that should be designated. For example, /LIST may also be given as /L, /LI, /LIS or /LIST.

Examples of Listing the Queues:

Example: .Q [115103,320571]/LIST
Function: List all jobs entries submitted by [115103,320571]. A typical printout is shown below:



Example: .Q [115103,320571]/FAST
Function: For the same queue inquiry, this switch will yield the following:

```
INP     115103,320571 TEST1   3287 *SZE      00:02:00   8K  0
LPTS10 115103,320571  B TEST  10116 *SZE          3
```

Example: .Q [115103,320571]/SUMMARY
Function: This command will yield a printout format giving the batch input queue inforamtion as well as the printer queue. A typical printout is as follows:

```
QUEUE SUMMARY AT 13:22:17 31-JUL-80  PITT DEC-1099/B 63A.44A

        EXECUTION QUEUES                        PRINT QUEUES (STD FORMS)
-------------------------------------------------------------------------

LIMIT  AV. AGE  OLDEST JOBS    TIME  *            OLDEST JOB
                                     * WHERE   <16 PGS    ALL JOBS  PGS
30SEC 00:00:00 00:00:00  0 00:00:00 *
 2MIN 00:00:00 00:00:00  0 00:00:00 *
10MIN 00:00:00 00:00:00  0 00:00:00 *
20MIN 00:00:00 00:00:00  0 00:00:00 *
30MIN 00:00:00 00:00:00  0 00:00:00 *
 1 HR 00:00:00 00:00:00  0 00:00:00 *
  LPQ 00:00:00 00:00:00  0 00:00:00 *
OTHER 00:01:31 00:01:31  1 00:02:00 *
```

Example: .Q INP:[115103,320571]/LIST
.SUBMIT [115103,320571]/LIST
Function: Both commands will print out the status of batch jobs submitted by [115103,320571].

Example: *.Q IPT: [115103,320571]/SUMMARY*
.PRINT [115103,320571]/SUMMARY
Function: Either one will print out the PPN's printer job status.

Example: *.Q INP:/LIST*
.SUBMIT/LIST
Function: Print out all batch jobs entries in the System.

Example: *.Q LPTS10:/LIST*
Function: Print out all printer jobs at station No. 10.

If the queued jobs have been completed when an inquiry is made, the System will return a message of: "The queues are empty."

Examples of changing the queues

Two switches may be used to change the queues: /MODIFY and /KILL. In a selective modification or deletion, the jobs must be identified by either the /SEQ:n switch or by the JOBNAME assigned. If /KILL is applied without a selective identification, every job fitting the specification will be deleted.

Example: *.Q LPT:=/KILL or .PRINT/KILL*
Function: Delete all printer jobs.

Example: *.Q LPTS10:=/KILL*
Function: Delete all printer jobs submitted to Station No. 10 printer.

Example: *.Q INP:TEST=/KILL or .SUBMIT TEST=/KILL*
Function: Delete a batch job whose jobname is TEST.

Example: *.Q LPT:=/SEQ:10116/KILL or .PRINT/SEQ:10116/KILL*
Function: Delete a print job which has been assigned a sequence number of 10116.

Example: *.Q LPT:=/SEQ:10116/MOD/BREAK:111111/LOC:6*
.PRINT /SEQ:10116/MOD/BREAK:111111/LOC:6
Function: Modify a print job, whose sequence number is 10116, by re-routing it to Station No.6 and programmer number 111111 as the receiver of the output.

Note: At the University of Pittsburgh, the command ".QUEUE INP:=" and ".SUBMIT" may be used for queue modification, but not for submitting batch jobs. The batch jobs are submitted by a Pitt-developed command of ".OPRSTK".

Example: *.Q INP:=/SEQ:3287/MOD/CORE:16*
.SUBMIT /SEQ:3287/MOD/CORE:16
Function: Change a batch job, whose sequence number is 3287, to a new core request (from whatever was before) of 16K.

The following examples show certain applications of the switches:

Example: *.Q NEWTON.FOR/LOC:10/COPIES:3/SPACING:DOUBLE*
Function: This command will submit a request to print 3 copies of NEWTON.FOR in the user's own disk area, double spaced, at the printer in Station No.10. Note that if the file is already double-spaced (with a blank line as every alternate line), the result with this command would be quadruple space.

Example: *.QUEUE FOR01.DAT/FILE:ASCII*
Function: A file with an extension of DAT is printed with an assumption it has the FORTRAN carriage control character format, because the /FILE:FORTRAN is the default switch for all files with the extension of DAT. Therefore, the FORTRAN data files are always prepared by reserving the first column of every line blnak except for carriage control characters. If a data file is not prepared this way, or if it has an unknown origin, it would be prudent to include the /FILE:ASCII swithc in the print request. Otherwise, for example, every time the printer encounters a line beginning with a "1" in column 1, the printer will take as an instruction to a new page.

Example: *.QUEUE PROB1.CDP/COPIES:2/DIS:REN*
Function: Punch two copies of PROB1.CDP in cards*, and remove the file from the owner's disk area immediately.

---

*Note to Pitt users: Pitt does card punch jobs outside the University. The punch job files are collected on tape once a week from the disk. A user should either keep his CDP file at least another week after he submits the punch request or use the /DISPOSE:RENAME switch.

Switch	Argument	Explanation	Default Argument
/CHECK		Same as /FAST, except that no PPN is required. Command format is simply "Q/CH".	
/FAST		List all jobs in the specified queue in a short format. Same as /LIST except no headings are printed.	
/LIST	:1	List the entries in the specified queue with a format as shown in the example shown in this Section.	:1
	:2	More detailed listing	
	:3	Most detailed listing	
/LOCATE	:n	Route the output to the specified printer at station No. n. Each station printer has a page limit, and long jobs can be printed only at certain printers. Also, only certain printers have lower case capability.	n=station number of RJE
/KILL		Delete an entry from a specified queue. If used with /SEQ:n switch or by the JOBNAME in the command, only the specified jobs are killed. Otherwise, all jobs in the specified queue are deleted.	
/MODIFY		Used in combination with another switch to change the value of that switch for a job already in the queue. The job to be modified must be identified either by a /SEQ:n switch or by the JOBNAME in the command. The specification of a previously submitted job may be modified by using the /MODIFY switch in conjunction with another QUEUE switch. To change the specification of a previously submitted batch job,use the following switches with the MODIFY switch: /CORE:n To change core limt to nK /OUTPUT:n To change the batch job OUTPUT parameter to n. /PAGES:n To change the page limit /PRIORITY:n To change job priority /TIME:hh:mm:ss To change CPU time limit	
/SUMMARY		To print out a summary of jobs in the queue.	

Table 8.5 Selected Queue Operation Switches of the QUEUE Command

Note: Underscored characters correspond to the short forms of the switches and the arguments.

Switch	Argument	Explanation	Default Argument
/AFTER		Job will not be processed until after the specified time:	
	:HHMM	After HHMM (using a 24-hour clock, HH=hour, and MM=minutes)	HHMM=0000
	:+HHMM	"HH" hours and "MM" minutes from now or later	
/BREAK	:N	Route the output to user whose programmer is N.	N=own PPN
/FORM	:LC	Route a print job to a printer that can print lower case characters.	
	STD	Print job on standard form, a default condition.	
/LIMIT	:n	Limit the output to the specified number of pages, cards, or feet of paper tape.	
/SEQ	:n	The sequence number identifies a particular job in a queue. This switch is normally used in combination with the /MODIFY or the /KILL switch to change or delete a seleted entry in the queue.	

Table 8.6 Selected General Queue Switches of the QUEUE Command

Note: Underscored characters correspond to the short forms of the switches and the arguments.

Example: .SUBMIT/SEQ:10116/MODIFY/CORE:32K
Function: To change a previously submitted batch job (identified by its sequence number) core request to 32K.

Example: .PRINT/MODIFY/LOC:6/FO:LC
Function: To re-route all printer jobs already submitted to location-6 and ask for lower-case printing.

Example: .PRINT /M/A:0000/LOC:3
Function: To re-route all printer jobs to printer-3 and do it after midnight.

Switch	Argument	Explanations	Default Argument
/COPIES	:N	To generate n copies of the output, but not larger than given in this table: Device — Maximum "n" LPT — 63 CDP — 3 PTP — 63 PLT — 3	:1
/DISPOSE	:DELETE	Files to remain in the owner's disk area, but to be deleted after processed.	:PRESERVE
	:PRESERVE	Files to remian in the owner's disk area, except those defaulting to /DISPOSE:RENAME.	
	:RENAME	Files to be immediately removed from the owner's disk area and placed in a system disk area, where it is neither accessible by the owner nor counted agains this disk quota. The file is deleted after spooling. This is the default for files with extensions of LST and TMP and if protection code is <0yz>, and extension one of these: CDP, LPT, PLT or PTP.	
/FILE	:ASCII	To specify that the file contains ASCII text. This is the default for all files except those with the extension of DAT.	
	:FORTRAN	To specify that the file contains in column-1 of each line a FORTRAN carriage control character. This is the default for all files with the extension of DAT. Applied only to printer jobs.	
/SPACING	:SINGLE	To print the file in single space.	:SINGLE
	:DOUBLE	To print the file in double space.	
	:TRIPLE	To print the file in triple space.	

Table 8.7 Selected File-Control Switches of the QUEUE Command

Note: Underscored characters correspond to the short forms of the switches and the arguments.

8.16 Operating System Command Locally Enhanced

While the Operating system is a part of the system software supplied by the computer vendor, local installations often modify certain commands to enhance their operations. At the University of Pittsburgh, there are a number of system commands enhanced to fit the local needs. Many take on different forms and keywords, such as DRIVE, QUEUE, and OPRSTK. These various enhancements have been presented also in this chapter. Users at other installations should consult with their own Computer Center staff and documentations with respect to the local system enhancement. In some cases, it is an upward-compatible enhancement so that a user not familiar with the enhancement may still use the command with the standard format and functions. One such command is the TYPE command. In the normal way, the TYPE command would run the PIP program and construct a command from the list supplied in the TYPE command. Thus no switches are allowed other than the standard default values of the PIP program.

At the University of Pittsburgh, a "TYPE" program is implemented and activated as a monitor command. The complete format for the command is:

```
.TYPE /global-switches  Output-spec = input list/local-switches
```

where: global-switch = switches applied to all files in the list
local-switch = switches applied to a specific file

The simplest form of the command would be with no switch, or:

```
.TYPE  input-list
```

Thus in its simplest form the TYPE command is the same as the DEC-10 monitor command TYPE.

The switches and their functions are tabulated in Table 8.8.

Example: .TYPE SYS:TYPE.HLP/PAGES:2-2/IND:4
Function: Type out Page 2 of the file SYS:TYPE.HLP with left margin at column 5.

Example: .TYPE NEWTON.FOR/COLUMNS:1-72/HEADER/PAUSE
Function: Type out columns 1-72 of the file NEWTON.FOR on user's terminal with a header. At the beginning of each page (including the first), the terminal beeps and pauses. This would allow a change of paper or allignment of typing position.

Example: .TYPE FCHART.LST/EMULATE:LABELS.CCT
Function: Many programs are designed for printer output only. One such program is the FORFLO program discussed in Chapter 4 (section 4.4). When a printer is not accessible, for example for a dial-up user, the output may be produced on the user's terminal by emulating his terminal as a printer. "LABEL.CCT" is the stored "printer carriage control tape." For this example, the FORFLO program uses CTRL-S to

Switch	Argument	Explanation	Default Argument
/ALIGN		Pause to allow alignment of paper before the first page. A carriage return resumes the typing.	
/PAUSE		Terminal will pause for paper alignment between every page. Carriage return resumes typing.	
/BLOCKS	:n-m :m	Process only the nth through mth blocks. Process first m blocks of the file. This switch is applicable only to disk files.	all blocks
/COLUMNS	:n-m :m	Proces only the nth through mth columns . Process the first m columns of each line.	all columns
/EMULATE	:filespec	This specifies that the output device is to emulate a line printer. The file specification is of the standard form, which contains information of a non-standard carriage control tape, or CCT. The detail content of the CCT file is given below.	standard settings
/FILE	:ASCII :FORTRAN	Treat the input file as ASCII text. Treat the input file having the first column a FORTRAN carriage control character.	:ASCII
/HEADER	:n	n=0: Omit the file processing information block. n=1: Type out the information block at the beginning of the file. n=2: Type out only the information block at the beginning of the file.	:0 if no switch; n=1 if argument missing
/INDENT	:n	Specify the left margin of output. Max n = 63. If column-1 has tab character in it, the indentation may be incorrect.	n=0
/LINES	:n-m :m	Process only the nth through mth lines. Process the first m lines of the file.	all lines
/PAGES	:n-m :m	Process only the nth through mth pages. Process the first m pages of the file.	all pages
/PRINT	:ARROW :ASCII :OCTAL :SUPPRESS	Type out control characters as up-arrow characters. Treat the file as straight ASCII file. Each word is converted to 12-digit octal number. Suppress all multiple-line spacing.	:ASCII
/TALLY		Type out the total number of pages of the file.	

Table 8.8 Switches for the Enhanced TYPE Command

suppress formfeed, and the user should have a file LABELS.CCT containing just one record "/DC3:1-66:1". The flow chart shown on page 145 was produced on a terminal in that way.

Example: .TYPE SAMPLE.DAT
Function: Output the file SAMPLE.DAT on user's terminal.

REFERENCES

1. DEC SYSTEM-10 OPERATING SYSTEM COMMAND MANUAL, DEC #DEC-10-OSCMA-A-D, Digital Equipment Corporation, Maynard, Massachusetts; 1974.

2. Update Notes to DEC SYSTEM-10 OPERATING SYSTEM COMMAND MANUAL, DEC #DEC-10-OSCMA-ADNI, Digital Equipment Corporation, Maynard, Massachusetts; 1976.

3. OPERATING SYSTEM COMMANDS, Reference Card, Computer Center, University of Pittsburgh, Pittsburgh, Pennsylvania; 1978.

4. GETTING STARTED WITH TOP-10 COMMANDS, Digital Equipment Corporation, Maynard, Massachusetts; 1976.

5. Various System HELP-files, Computer Center, University of Pittsburgh, Pittsburgh, Pennsylvania; 1980.

6. The QUEUE COMMAND, DEC-10 Notes, Computer Center, University of Pittsburgh, Pittsburgh, Pennsylvania; October, 1980.

CHAPTER 9

MULTIPROGRAM BATCH

INTRODUCTION

9.1 Introduction

The conventional batch operation of a computer will dedicate all system resources to a current job. When this job is completed, the system resources are then reassigned to the next job.

The DEC System-10 is a multiprogramming system. It allows multiple independent jobs to reside in the core simultaneously and to run concurrently, by sharing the core and by switching the central processor from one job to another when the processor becomes available. Using this approach, the DEC batch enables the execution of up to 14 jobs concurrently with time-sharing jobs*. Thus the multiprogram batch (MPB) is a hybrid mode of operation between the conventional batch and the conventional time-sharing modes.

As a hybrid operation, the MPB operations often retain the advantages and the disadvantages of both modes. Its chief advantages are:

(1) Computer resources are used more efficiently to increase throughput, namely, more users served in a given length of time.

(2) Programs and data can be prepared on key punches, which are less in demand than the terminals. The prepared card deck serves as a permanent storage in user's possession unrestricted by disk allocation quota and unthreatened by the prospect of a system crash.

(3) Jobs may be run by an operator, or by user using a self-service card reader at a remote job entry station. Output may be retrieved later at the user's convenience.

(4) Batch jobs have larger core allocations. Some large and long jobs can only be run in the batch mode.

---

*Five jobs only from the user's standpoint.

On the other hand, the major disadvantage of the MPB mode is the loss of interaction between the user and the computer.

Since a job can normally be run on DEC-10 either as a batch job or as a time-sharing job, a judicious choice should be made by the user to maximize the advantages and to minimize the disadvantages. For example, a program debugged in the time-sharing mode and run in the batch mode would combine the best of two worlds.

In general, the following types of jobs are best suited for the batch mode:

(1) Production runs of a program already debugged.

(2) Large and long jobs.

(3) Jobs requiring large amount of input data and/or producing large amount of output data.

(4) Jobs requiring no interaction between the user and the computer at the execution time.

(5) Jobs of users who have difficulties to gain access to time-sharing facilities.

9.2 BATCH Software System

For the purpose of illustrating the MPB system, a very brief description of the BATCH software system will be given here.

There are four major components in the MPB software system. They are the Stacker, the Queue Manager, the Batch Controller, and the Output Spooler. Each is a service program, and their interaction and control functions are shown in Figure 9.1.

The Stacker program performs the service of reading the input from the input devices and entering the job into a batch queue. Since each installation of DEC-10 system has variation of input device configuration, the Stacker is primarily oriented toward the card reader input, but it allows jobs to be entered from any input device that is code-compatible. In some installations, this would include the terminal and/or the disk*. Once the information is entered, appropriate data files are set up, monitor commands are generated that will later execute the job, and a job log file is set up to record the case history of the job. The log file is a part of the standard output a user receives after his batch job is completed, whether successful or not.

The Queue Manager program schedules the job, both for the jobs in the batch controller's input queue and the output job queue. When a job request reaches the Queue Manager, its priority is computed by a formula based on such job parameters as the CPU time requested, core size requested, etc. The priority is further upgraded from time to time as the job ages in the queue. When the batch input job is completed, the Manager sends forward a request of

---

*At the University of Pittsburgh installation, a modified Stacker called the Operation Stacker (OPRSTK) is implemented to replace the DEC Stacker in order to unify all input/output device queues under a single queuing control.

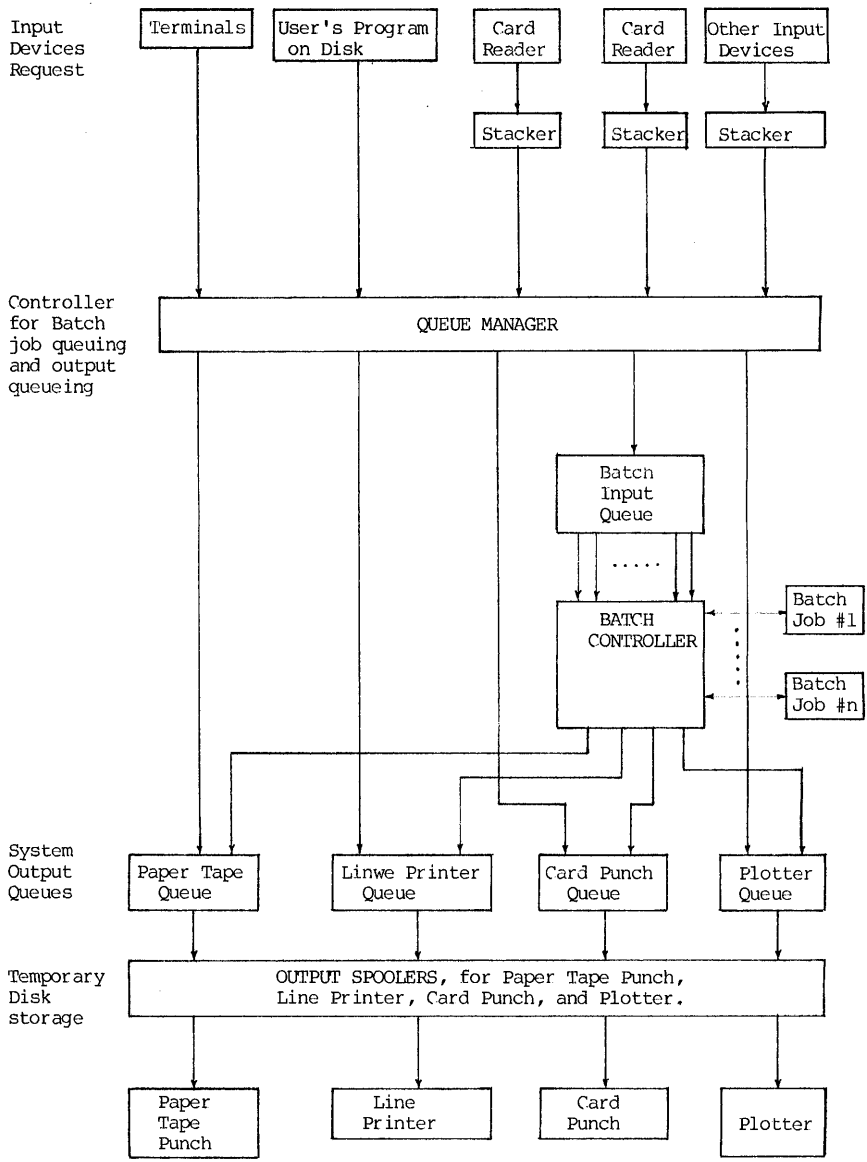


Figure 9.1 BATCH Software System

scheduling to the Output Queue. After the output is completed, the Queue Manager scratches the job entry from the queue.

The Batch Controller program processes the batch jobs by passing the monitor commands, generated by the Stacker, onto the executive system for action. Here one should observe that the batch jobs processed by the Batch Controller are not distinguishable regarding the sources of these jobs. Jobs originated from the remote terminals, from the card readers or from any code-compatible input device will appear the same to the Batch Controller. Therefore, the time-sharing and the batch jobs can share the same operating system and the non-resident software commands, and it is unnecessary for a user to learn two different sets of command languages, one for time-sharing and one for batch. Also, so long as the input codes are compatible, any input device may submit a batch job. However, since a user can sign on the System only with the card reader or with the terminal, our discussion of batch jobs will be limited to these two sources only.

The Output Spooler program allows the output to be stored temporarily on a system disk at a queue priority assigned by the Queue Manager. When the specified output device becomes available, the appropriate spooling program processes the output job. Note in Figure 9.1, the batch system output devices do not include the remote terminals, and hence the terminal users, while it is possible for them to submit batch jobs at the terminals, will not receive the job output at their terminals. They must obtain the batch output at the System output devices.

9.3 Procedure of Running a Batch Job

At this point, we shall assume that the program has been written and prepared either as a card deck or as a stored file, the data assembled and prepared, and the user is ready. The procedure of running a batch job includes the following steps:

(1) First, a control file should be made up, which contains the program, the data, the monitor commands and/or the BATCH commands in a proper composition and sequence. The control file made up can be either a complete deck of cards or a stored disk file.

(2) Secondly, submit the batch job for the run. This can be done either through a card reader (for control file punched on cards) or through a terminal (for control file stored on disk).

(3) Thirdly, retrieve the output results and interpret the results. If there are errors, detect and correct them, and resubmit.

These procedures will now be discussed in details next.

CONTROL FILE

9.4 Batch Control Commands

The batch control commands are commands whose functions are recognized by the Stacker in the Batch software system so that appropriate commands can be generated. These control commands are all characterized by a dollar sign ($) in the first column and the command keyword starting at the second column. A selected set of batch control commands is presented next.

The general format of a batch control command is:

```
$KEYWORD /sw1 /sw2 ... /swn
```

where "$" = the command symbol; must be in the first column;

KEYWORD = the batch command keyword; must start from the second column;

/swk = a switch or option specifying certain parameters (for example, core size, CPU time limit, paper page limit, etc.)

If a particular switch is not used in the command, the system automatically assigns a prescribed and safe option for the command. For example, a user may use the page limit (of the printer output) option to specify the limit up to 999 pages. If that option is not specified in the command, the system automatically assigns 15 as the page limit. Such automatically assigned option is called the default condition of the switch.

The control commands presented here will often be referred to as "command cards," even if they need not be always in card forms physically. Since they always have a dollar sign ($) in the first columns, they are also called the $-cards.

9.5 Sign-On Batch Control Commands

These command will generate the monitor command LOGIN with specifications of job requirements. Local installations often have variations and options of these commands. Users should confirm with the personnel at their local installation.

(1) $SEQUENCE Card This command will specify a unique sequence number for the job. It may or may not be required at a local installation*. If it is required, it must always be the first card in the control file.

(2) $JOB Card This command will generate the LOGIN command and make specification and limits of job requirements. Its form is:

```
$JOB  JBNAME [m,n] /sw1 /sw2 ... /swn
```

*At the University of Pittsburgh, the $SEQUENCE card is not needed.

where JBNAME = the optional job name given by the user. If this is omitted, the System assigns an arbitrary name such as JOBAA. The purpose of the JBNAME of a job is for its identification.

[m,n] = user's PPN; must not be omitted.
When the $JOB card is prepared by an IBM key punch, which does not have the "[" and the "]" punches, the PPN may be punched either as ¢m,n! or as (m,n).

/swk = JOB switches. A list of selected JOB switches is shown in Table 9.1.

If the default conditions are satisfactory in an application, the simplest form of the $JOB card could be simply:

```
$JOB [m,n]
```

which will tell BATCH: (1) to login with a PPN of [m,n], (2) to require no service of card punch, paper tape punch, or plotter, (3) to request the regular amount of core size, regular level of priority, and regular CPU time limit, and (4) to return the output at the station where the BATCH run is submitted. In the meantime, a job name may be arbitrarily assigned by the System.

At the University of Pittsburgh, a short form of JOB card is permitted in the form of:

```
$JOB JBNAME [m,n] (time,pages,core,cards,feet)
```

The parameters of switches are given inside the parentheses and must be given in that specific order. If the default value is used, that parameter may simply be left blank, but the comma must be retained, unless it is a trailing comma. Additonal switches may be appended after the parentheses are closed.

Example: The two following JOB cards are equivalent:
```
$JOB NEWTON[115103,320571]/TIME:1:00/CORE:12K
$JOB NEWTON[115103,320571](1:00,,12K)
```

The two following JOB cards are equivalent:
```
$JOB [115103,320571]/CORE:12K/PRIORITY:0
$JOB [115103,320571](,,12K)/PRIORITY:0
```

When a batch job is submitted from a time-sharing terminal, the PPN of the user is already known to the System. Therefore, such jobs are allowed to have PPN containing a wild card construction such as [*,320571], [115103,*], or [*,*]. The advantage of such a construction is that one copy of such control file need be stored and it can be shared by many projects for the same user, or many users in the same project, or by anybody. When such a job is submitted, the system will substitute the known PPN for the "*" part of the JOB card.

Many of the JOB card switch parameters pertain to the estimated maximum usage of system resources, such as the CPU time, the core storage, and the output volumns that are required for the job. The user should try to estimate with a margin as close as possible above all of his requirements. If the requirements are under-estimated, either the job may not run, or the output may be cut off before it is completed. On the other hand, although overestimates will not cause a waste in the System resources (because they are allocated only when actually needed in execution), it will cause a drop of the job ranking in the Batch queue, and the turn-around time will suffer. A useful guide for a reasonable estimate is to look at the KJOB printouts of past simialr jobs, to

Switch	Argument	Explanation	Default Argument
/AFTER	:HHMM :+MM :DD-MM-YY HHMM	Job will not be processed until after the specified time: After HHMM (using a 24-hour clock, HH=Hour, and MM=minutes) "MM" minutes later Process the job after HHMM on the specified day. For example, 8-AUG-80 1350 means 1:50PM, August 8, 1980.	HHMM=0000
/CORE	:nK :n :nP	Estimated maximum core request required for the job: core in K-words core in K-words (1 K = 1024 words) core in P-pages (1 P = 512 words)	Consult your local rules.
/LOCATE	:Snn :nn :name	Route the printed output to a specified RJE station. nn = station number name = station name	Same Snn as RJE
/OUTPUT	:n	Control the automatic queuing of the output at the end of a job. Argument "n" is listed below: n — File Queued 0 — no queuing 1 — only LOG file pritned 2 — Log file plus spooled output 3 — Log file, spooled output, *.LST 4 — Same as 3 5 — All except LOG file	n=4
/PAGES	:nn	Estimated maximum number of printed pages	
/PRIORITY	:n	Set the priority level of the job to "n": n=3 for high priority (HPQ) n=2 for normal priority (NPQ) n=1 for low priority (LPQ) at lower rate n=0 for no priority (NPQ) at bargain rate	n=2
/SITGO		Job is to be run under the SITGO batch.	
/TIME	:HH:MM:SS	Estimated maximumum CPU time required	30 sec
/UNIQUE	:n	n=0: Job may be run simultaneously with other jobs of the same PPN. n=1: One job per PPN at a time.	n=1

Table 9.1 A List of Selected $JOB Card Switches

see how much CPU time was used, and to see what was the maximum core area used in that job (called the HWM core, or the "high-water-mark" core), and then add 10%.

(3) $PASSWORD card This card has a format of:

```
$PASSWORD password
```

where "password" is the user's password consisting of zero to six characters that have been stored with his PPN. The $PASSWORD card must follow immediately after the $JOB card, and these two together identify the batch user to the BATCH processor and initiate the creation of necessary temporary files for the job. If the password given is not valid, the job is terminated right there.

When a batch job is submitted from a terminal, the $PASSWORD card is not needed. The mere fact that the user is operating the terminal at the time is proof enough that he has a valid password. In this case, the system simply retrieves the stored password of the user.

Thus the simplest version of job sign-on at the beginning of each batch job is shown below:

```
$JOB [m,n]
$PASSWORD password
```

or

```
$SEQ
$JOB [m,n]
$PASSWORD password
```

where the "$SEQ" card is optional depending on the local installation conditions.

9.6 Sign-Off Card, $EOJ

The sign-off card is placed at the end of a Batch deck to tell BATCH that it has reached the end of a job. The design of the sing-off card varies with local installations. Two differenet cards are described below:

(1) The End-of-File Card This is the standard sign-off card recognized by the DEC-10 BATCH system. It is a card with punches in columns 1 and 80 in rows 12,11,0,1 and rows 6,7,8,9 leaving rows 2,3,4,5 blank. While it is possible to produce these punches on a key punch, it is quite difficult to reproduce that card as a record on a disk file, and thus causing problem in indicating the end-of-job when submitting the job through a terminal. This sign-off procedure is not used at the University of Pittsburgh.

(2) The $EOJ Card The $EOJ card is implemented to replace the end-of-file card at the University of Pittsburgh installation. It will generate the monitor command KJOB and sign off the user with appropriate printout of usage data. It should be physically the last card in every BATCH deck. If this card is omitted, the end-of-job may still be determined by the presence of $JOB card of a following job, assuming that the user of the following job provides a correctly prepared $JOB card. If there is no job following, or if the following job card has error in it, difficulties may arise. Therefore, although it may not be always needed, it is prudent to use two $EOJ cards at the end, just in case there should be any read-error by the card reader.

9.7 The End-of-Deck Card, $EOD

During the batch run, a number of batch control commands will tell BATCH to copy all cards following the command card into a disk file. Such copying of cards may be terminated only on two conditions: (1) when another $-card is encountered, or (2) when an end-of-deck card $EOD is encountered. The format for the latter is:

```
$EOD
```

As we shall see later, not every command card in the Batch deck is a $-card, it will be prudent to end every program and every group of data with a $EOD card, even though it may often be unnecessary. If each of the $card followed by the program or data to be copied ends with a $EOD card, this becomes a stand-alone module in the Batch deck. In this way, when the modules are moved or removed, or when additonal commands are inserted into the Batch deck at a later time, one can always be certain that these modules in the Batch deck have already been properly terminated.

9.8 Batch Control Commands for Disk Storage

The Batch control command $DECK placed in front of a deck of program and/or data cards will tell BATCH to copy it into a disk file, and do not delete it at the conclusion of the batch run. Its form is:

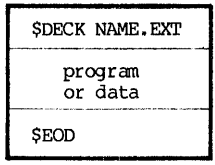
```
$DECK NAME.EXT/switches
```

Switches for the $DECK card are the same as those for the $DATA card, except for the IGNHOL switch, and they are listed in Table 9.2:

Switch	Argument	Explanations	Default Argument
/WIDTH	:N	To read the first n columns of each card as the data columns. (n<80)	n=80
/SUPPRESS	:ON :OFF	To suppress the trailing blank of each card. To read the trailing blanks of each data card as data columns.	:ON
/LINEBLOCK		To "lineblock" a data file so that it may be read by a FORTRAN F40 program. Lineblocking is not required in a FORTRAN-10 program.	
/IGNHOL		This is a switch for the $DECK card only. It is used to ignore the Hollerith errors in a deck of cards. Any column containing a Hollerith error is replaced by a backslash (\) before being stored on disk. The disk file can then be edited to correct the errors.	

Table 9.2 Switches for the $DECK and the $DATA Cards

The disk storage module in a batch job deck has the following composition:



9.9 Batch Control Commands for Compiling and Execution

When the BATCH processes a FORTRAN job, it will assign a FORTRAN compiler and set aside disk area for temporary storage for these files: (1) the source programs (programs in source language such as FORTRAN) and their compiled binary files (with extensions of REL), (2) REL files that are already on disk, (3) data cards. When all files are stored as temporary files, the programs are then executed. The Batch control commands presented below will generate necessary monitor commands that will perform these tasks.

(1) Compiling command for source language programs Batch commands in this group will tell BATCH to generate a monitor command COMPILE to process the source program deck that follows the command card. Since there are several language processors that can be processed by the COMPILE command in DEC System-10 (ALGOL, COBOL, FORTRAN, MACRO, BLISS and SIMULA), six Batch control commands for compiling are provided. They are $ALGOL, $COBOL, $FORTRAN, $MACRO, $BLISS and $SIMULA. Although the discussion in this section are directed to the command $FORTRAN, one may extrapolate the discussions to ALGOL, COBOL or MACRO since they share the similar formats, similar functions and the same switches.

The format of the $FORTRAN card is:

```
$FORTRAN  NAME.EXT/switches
```

where NAME.EXT is the name of the temporary file to be created to store the FORTRAN cards that follow. When the $FORTRAN card is followed by a FORTRAN source program deck, three events will take place: (a) The FORTRAN deck is copied onto the disk and named as "NAME.EXT" as designated in the $FORTRAN card; (b) the source program is compiled; and (c) the compiled result is stored on disk as NAME.REL. In the meantime, a temporary listing file is also prepared. The name NAME.EXT can often be omitted, and BATCH will create an arbitrary name for the program file, such as DECKAA.FOR, DECKAB.FOR, etc. All files stored are temporary, and they will be deleted at the end of the Batch run. Threrefore, $FORTRAN card should not be used if the user wishes to compile and store the files.

The simplest version of the $FORTRAN card is simply:

```
$FORTRAN
```

which tells the BATCH processor: (1) to copy the FORTRAN program into a temporary file, (2) to assign it a name (DECKAA.FOR), (3) to read 72 columns of each card following the $FORTRAN card as a FORTRAN program, (5) no CREF lisitng, (5) to produce a listing of the program when the job is done, and (6) to use the FORTRAN-10 compiler. Although simple in form, it is the most useful form of the $FORTRAN card.

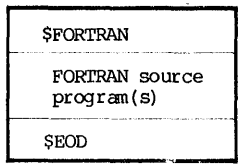
The following is a list of selected switches for the $FORTRAN card:

Switch	Argument	Explanations	Default Argument
/WIDTH	:n	To read only the first n columns of each card. (n<72)	n=72
/CREF		To create a cross-reference listing of the FORTRAN program when compiled.	No CREF listing
/NOLIST		No listing of the program is to be prepared.	Listing
/F40		Use the FORTRAN F40 compiler.	/F10
/F10		Use the FORTRAN-10 compiler, a standard compiler at Pitt.	

Table 9.3 Switches for the $FORTRAN Card

Two limitations of the $FORTRAN card should be noted. One is that it will list, compile but not execute the program. The other is that although files will be created to store the source programs and the compiled binary REL programs, they are all temporary and will be deleted at the end of the job. If one wishes to keep the FORTRAN or its REL files as permanent files for later use, $FORTRAN would be a wrong command to use. Instead, one should use the $DECK card to store, and the monitor command COMPILE to compile and create the REL files.

The general composition of a compiling module in a Batch deck is as follows:



When a FORTRAN program consists of a main program and one or more subprograms, it can be compiled either together in a deck as one module, or compiled separately in individual decks, each requiring one such module. As mentioned before, the $EOD card may be omitted if the first card of the next module is a $card. However, if the deck is modified in the future and a non-$card is inserted at the end without a $EOD terminating the FORTRAN module, the inserted command will be interpreted as a part of the FORTRAN program and causes an error termination of job when the program is compiled. Therefore, it is prudent to provide always a $EOD card at the end of the module as a standard practice.

(2) Inclusion command for other REL files Not all components in a FORTRAN job are source programs. Many are binary relocatable REL files, which are already compiled from certain source programs and stored on disk. Sometimes, it is preferable to use the REL file because it will save time not to re-compile. Other times, the source language programs may not even be available

except the REL files. Since $FORTRAN card can only handle programs in source language, another command called $INCLUDE should be used to include the REL files needed for later execution. The function of $INCLUDE card is to find the specified REL files and copy them onto the disk as temporary files. It has a form of:

```
$INCLUDE  NAME.REL[m,n]/switches
```

where NAME.REL is a list of REL files to be included. The PPN follows the conventional rule and is omitted if it is user's own PPN. The switches of the $INCLUDE card are shown on Table 9.4.

Switch	Explanations
/LIBRARY	The REL file referenced usually contains many subprograms. Not all of them are needed in the job. The LIBRARY-switch will specify a search mode and include only those subprograms called by the main program. Therefore, the $INCLUDE card with the LIBRARY switch should be placed after the program calling the subprogram.
/PROGLIB	The REL file specified by this switch is stored in the Program Library (PRG:). The two following commands are equivalent: $INCLUDE NAME.REL/PROGLIB $INCLUDE PRG:NAME.REL
/SYSTEM	The REL file specified by this switch is stored in the System Library (SYS:).

Table 9.4 Switches for the $INCLUDE Card

(3) Execution command The Batch control command $DATA is placed in front of the data cards to tell BATCH to copy the data into a disk file and to insert a monitor command EXECUTE into the control file. When the job is run, any FORTRAN program or programs that have been entered before the $DATA card, and any REL files that were included by the $INCLUDE card will be executed. Since the $DATA card generate an EXECUTE command, it is required for execution even if the program run needs no "data". In such a case, a $DATA alone with no data card following should be used for execution.

The general form of the $DATA card is as follows:

```
$DATA  XXX.CDR/switches
```

where XXX.CDR is the optional 3-character file name specified by the user. If this is omitted, BATCH creates a unique name for the data file, for example, QAA.CDR. The switches for the $DATA card are the same as those of $DECK card as shown in Table 9.2, with the exception of the IGNHOL switch.

The simplest version of the $DATA card is simply "$DATA", which tells BATCH to copy the data into a file, to assign a unique name with an extension of CDR such as QAA.CDR, to read all 80 columns of each card as data, to delete the trailing blanks when copying them into a file, and then to execute.

Each $DATA card generates only one EXECUTE command. Hence, if you wish to execute a program several times with different sets of data, several $DATA cards are required, each followed by the appropriate set of data cards. Thus, the composition of an execution module in a Batch deck is shown below:

```
$DATA
  data cards
$EOD
```

A $DATA module followed by another $DATA module without intervening $FORTRAN will load and execute the same program again. However, if a $FORTRAN card follows a previous $DATA card, this terminates the previous program run and starts a new program run. Any subsequent $DATA will execute the new program. Therefore, the $DATA card should be placed in the Batch deck at such a place where all required FORTRAN programs have been compiled and all required REL files have been included.

9.10 A Summary of Batch Deck Modules

The Batch control commands presented are now summarized in the form of deck modules:

```
$JOB JBNAME[m,n]/sw
$PASSWORD password
```
Sign-On module

```
$EOJ
```
Sign-Off module

```
$DECK NAME.EXT/sw
  program or data
$EOD
```
Disk-Storage module

```
$FORTRAN NAME.EXT/sw
  FORTRAN programs
$EOD
```
Compiling module

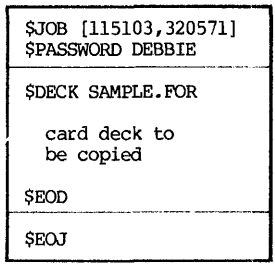
```
$INCLUDE NAME.REL/sw
```
Inclusion module

```
$DATA XXX.CDR/sw
  data cards
$EOD
```
Execution module

Construction of a control file then becomes a task of assembling these modules in a proper order with appropriate contents. Examples are now presented next.

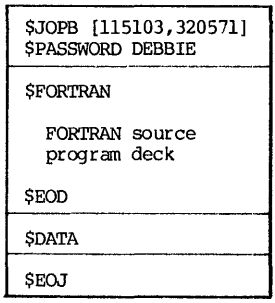
In the following examples, we shall assume that the user has a PPN of [115103,320571] and his password is DEBBIE. Previous remarks on $EOD cards also apply here.

Example 1: Copy a card deck (program) into a disk file and name it as SAMPLE.FOR.



A card deck, consisting of cards in the order as shown is assembled. When this deck is put through a card reader, a Batch job of copying a program deck onto the disk is submitted. At the conclusion of the job, there will be a disk file named SAMPLE.FOR in the disk area of the user [115103,320571].

Example 2: List, compile and execute a FORTRAN program which need no data:



In this example, although several temporary files are created during the job, no permanent file remains after the job.

Example 3: List, compile and execute a FORTRAN program with two sets of data. Note large amount of core and printout pages requested in the $JOB card:

$JOB [115103,320571]/CORE:24K/TIME:2:00/PAGES:100 $PASSWORD DEBBIE	The sign-on module
$FORTRAN FORTRAN program deck $EOD	The compiling and listing module
$DATA Data deck #1 $EOD	Execution module, first run.
$DATA Data deck #2 $EOD	Execution module, second run.
$EOJ	The sign-off module

Example 4: Execute a FORTRAN program with several separate decks of subprograms. The main program and the subprograms are MAIN.FOR, SUB1.FOR and SUB2.FOR respectively. In addition, the program calls for subroutines that are a part of a library package ENG:SUBSET.REL.

$JOB [115103,320571]/CORE:12K/TIME:2:00/PRIORITY:0 $PASSWORD DEBBIE	The sign-on module
$FORTRAN MAIN.FOR MAIN.FOR deck $EOD	Listing and compiling module
$FORTRAN SUB1.FOR SUB1.FOR deck $EOD	Listing and compiling module
$FORTRAN SUB2.FOR SUB2.FOR deck $EOD	Listing and compiling module
$INCLUDE ENG:SUBSET.REL/LIB	Inclusion module
$DATA data deck	Execution module
$EOJ	Sign-off module

Example 5: The major functions of several batch control commands are to store the decks temporarily and to issue the commands COMPILE or EXECUTE. If the files are already stored on disk, these $cards may be replaced by suitable monitor commands. In such cases, the command card should begin with a period (.) in the first column. Moreover, any monitor command, with the exception of those irrelevant for batch operation such as the SEND comamnd, may be included in the control file. Examples 2, 3 and 4 are now repeated below using monitor commands replacing some of the $cards:

```
$JOB[115103,320571]
$PASSWORD DEBBIE
.UPDATE SAMPLE.FOR
  FORTRAN source
  program deck
$$END
.EXECUTE SAMPLE.FOR
$EOJ
```

Alternate Batch Deck for Example 2

```
$JOB[115103,320571]/switches
$PASSWORD DEBBIE
.UPDATE SAMPEL.FOR
  FORTRAN source
  program deck
 $$END
.EXECUTE SAMPLE.FOR
  Data deck #1
$EOJ
```

Alternate Batch Deck for Example 3

```
$JOB[115103,320571]/CORE:12K/TIME:2:00/PRIORITY:0
$PASSWORD DEBBIE
$DECK PRGM.FOR
   MAIN.FOR deck
   SUB1.FOR deck
   SUB2.FOR deck
$EOD
.EXECUTE PRGM.FOR, ENG:SUBSET.REL/LIB
  Data deck
$EOJ
```

Alternate Batch Deck for Example 4

The main difference between these alternate Batch decks from the original solutions given is that the FORTRAN files and their compiled REL files are now all permanently stored in the user's disk area.

The Batch Controller controls all jobs that enter the BATCH system. It reads each line in the control file and determines its destination by interpreting the character in the first two columns. The interpretations by BATCH for these characters are tabulated in Table 9.5.

Thus, by combining the monitor commands and the CUSP commands, a Batch deck may be constructed without any special Batch control card except those to sing-on and to sign-off. Examples below illustrate this flexibility:

Column 1	Column 2	Intespretation	Example
$	Alphabet	Batch control command (or $card)	$JOB - batch command $JOB
$	Numeric or special character	Data beginning from column 1	$123.95 - interpreted as a data
$	$	Column 1 is suppressed. The line is taken as data beginning from column 2. Used to represent data of the form $-alphabet.	$$JOB IS A BATCH COMMAND - interpreted as "$JOB IS A BATCH COMMAND", a string data consisting of characters.
.	Alphabet	Monitor command	.R UPDATE - interpreted as a monitor command
.	Non-numeric	Data including column 1	.1095 - interpreted as a data
*		CUSP level command	.R BASIC To run a BASIC *OLD,NEWTON program named *RUN NEWTON.BAS
= ; %		Comment	;CONTINUE
^	Alphabet	Control-character	^C interpreted as Control-C
^	^	Multiple ^ in succession counted	^^^^C interpreted as Control-C

Table 9.5 A Summary of Line Interpretation by the Batch Controller

Example 6: Execute a FORTRAN NEWTON.FOR which is already stored on disk.

```
$JOB[115103,320571]
$PASSWORD DEBBIE
.EXECUTE NEWTON.FOR
     (Formatted data required by NEWTON.FOR)
$EOJ
```

Example 7: Enter a PIL program, run it and save it as NEWTON.PIL

```
$JOB[115103,320571]
$PASSWORD DEBBIE
.R PIL
     (PIL program deck)
     (Program begins with Part 1)
DO PART 1
     (Data deck)
SAVE AS "NEWTON", ALL PARTS, ALL FORMS
$EOJ
```

9.11 Batch Control Commands for Error Recovery

Normally, when an error occurs in the job, BATCH will report the error on the log file and terminate the job. Error recovery means to provide the user with an alternative should the error occurs. The formats of the error recovery control commands are:

```
$ERROR   statement

$NOERROR statement
```

The commands are interpreted this way by the BATCH processor: If the command keyword (ERROR or NOERROR) is "true", execute the "statement" following the command. Otherwise, execute the next line in the control file. The "statement" in the command is another command to the monitor, to a system program or a special BATCH command such as .GOTO or .BACKTO as an error branching command.

The Batch commands of .GOTO and .BACKTO have the form:

```
.GOTO statement label

.BACKTO statement label
```

where the "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 labeling a line.

The .GOTO command will transfer the control of BATCH forward to the reference line which contains the label. The .BACKTO command does the similar thing except it transfer the control of BATCH backward to a reference line containing the label. If the search for the reference line with the specified

label is not successful, the BATCH terminates the job.

Example: The new version of an old program OLD.FOR has been prepared as NEW.FOR. Both are stored on disk. The user now wishes to run the batch in this manner: If the new version works, use the new version. If the new version still has bugs, then use the old version.

The problem logic may be represented as a flow chart as shown in Figure 9.2. Each of the flow chart blocks is numbered, and these numbered blocks correspond directly with the modules of the Batch deck assembled as shown in Figure 9.3.

9.12 Miscellaneous Topics in Batch Control Commands

(1) Batch control commands developed at Pitt At each local installation, often additional $cards are created to handle local needs, such as a particularly heavy demand of certain "canned" programs, or some special procedures instituted at a local installation. A group of $cards created at the University of Pittsburgh are presented here:

$Card	Explanations and Examples
$BMD (xxx)/switches	The BMD (BioMeDical Computer Programs) is a canned program package developed originally at UCLA. Readers are referred to the References for more details. "xxx" is the last three characters of the name of the desired BMD program. Example: $JOB [115103,320571]/CORE:32K/TIME:2:00 $PASSWORD DEBBIE $BMD (07B) BMD control cards and input data cards $EOJ
$CSMP/switches	The CSMP (Continuous System Modeling Program) is a canned program originally developed by IBM. This $card is placed immediately before the CSMP source deck. Example: $JOB [115103,320571]/CORE:28K/TIME:2:00 $PASSWORD DEBBIE $CSMP CSMP source deck $EOJ
$DRIVES DEV(n)...	To request allocation of tape drives. "DEV" is one of the following tape drives: MT7 7-track magtape drive MT8 9-track magtape drive (800/1600 BPI) MT9 9-track magtape drive (1600/6250 BPI) DTA DECtape drive n number of drives required for the job. If n is 1, it may be omitted along with the parentheses This $card may be placed anywhere after the $PASSWORD card. It is required for batch jobs that use tapes.
$EOJ	It must be physically the last card in the Batch deck. See Section 9.6 for details.

$INCLUDE	It directs BATCH to include some existing REL files when the program is executed. See more details in Section 9.9.
$RUN DEV:NAME.EXE[m,n]/switches	This card is placed directly in front of data cards when running a program previously saved as an EXE file, created by a SAVE command. Example: The core image of a program has been previously saved as SAMPLE.EXE $JOB [115103,320571]/CORE:15K/TIME:2:00 $PASSWORD DEBBIE $RUN SAMPLE Input data deck for the program $EOJ
$SPSS/switches	This switch is used to run the SPSS (Statistical Package for the Social Sciences) program, and is placed immediately before the control cards and input data cards. Example: $JOB [115103,320571]/CORE:28K/TIME:2:00 $PASSWORD DEBBIE $SPSS SPSS control cards and data cards $EOJ

(2) A summary of switches for the $cards The $cards have many options by using the form of switches. These have been described previously in this chapter accompanying the associated $card. Options need not always be exercised. When a switch available is not specified in a $card, the System assigns a "default value" for the option. The default conditions of all switches for the Batch Control Commands are tabulated on Table 9.6.

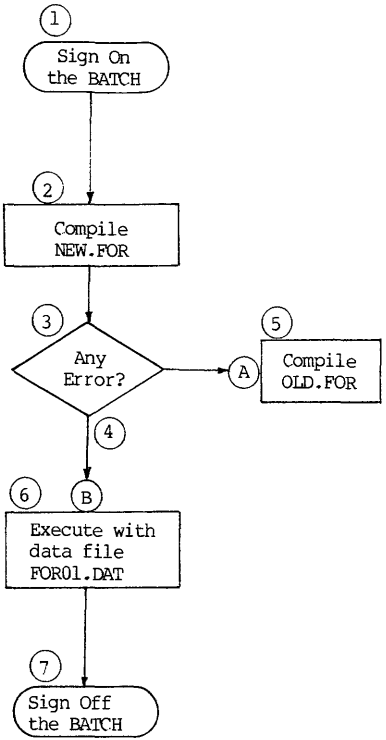


Figure 9.2 Flow Chart Logic

1	$JOB [115103,320571] $PASSWORD DEBBIE
2	$FORTRAN NEW.FOR program deck of NEW.FOR $EOD
3	$ERROR .GOTO A
4	.GOTO B
5	A::; CONTINUE $FORTRAN OLD.FOR prgram deck of OLD.FOR $EOD
6	B::; CONTINUE $DATA FOR01.DAT data deck $EOD
7	$EOJ

Figure 9.3 Batch Run Deck

$Card Switch	$LANGUAGE	$DATA	$DECK	$BMD	$CSMP	$INCLUDE	$RUN	$SPSS
/026 /D026 /ASCII	X X D	X X D	X X D	X X D	X X D		X X D	X X D
/LINEBLOCK	X	D	X	D	D		D	D
/CREF	X							
/WIDTH:n	note 1	n=80	n=80	n=80	n=80		n=80	n=80
/SUPPRESS:XX	note 2	XX=OFF	XX=ON	XX=OFF	XX=OFF		XX=OFF	XX=OFF
/LIST /NOLIST	D X							
/SYSTEM						X	X	
/PROGLIB						X	X	
/LIBRARY						X		

Legend: D = The option chosen by the $card if a switch is omitted.
X = The switch is valid on that card type.
Blank = the switch is invalid on that card type, and the job will be cancelled.

Note 1: n=80 for $ALGOL and $MACRO; n=72 for $COBOL and $FORTRAN
Note 2: XX=OFF for $FORTRAN; XX=ON for $ALGOL, $COBOL and $MACRO

Table 9.6 Default Conditions for the Selected Switches of Batch Control Commands

SUBMITTING A BATCH JOB

9.13 Submitting Batch Jobs in Cards

Submitting Batch jobs in cards is the most common procedure of running the batch jobs. After the Batch deck is prepared and assembled, the deck is submitted for reading into the Batch queue by a card reader. In some installations, the deck is submitted to the personnel of the Computer Center. In other installations, the users operate self-service card readers to read in their jobs at a remote job entry station.

9.14 Submitting Batch Jobs from a Terminal

To submit a Batch job from a terminal, the user must first prepare a control file and store it on disk. The details of how to prepare a control file for a job have been given in the earlier part of this chapter. From that point on, there are several ways to run the Batch jobs from the terminal. These various ways are not always all available at a local installation. It is necessary for the user to confirm which way is used at his installation.

(1) By using monitor commands SUBMIT or QUEUE INP: The monitor commands SUBMIT and QUEUE INP: are used to place entries into the input queue for the Batch system. Their formats are as follows:*

```
SUBMIT JBNAME = NAME.CTL, log file
QUEUE INP: JBNAME = NAME.CTL, log file
```

where JBNAME = name of the job being entered into the queue.

NAME.CTL= name of the control file. This file contains all monitor-level and Batch control commands for processing by the Batch Controller.

log file= name of the log file. This file is used by the Batch Controller to record the case history of the job processing.

(2) By using specially implemented service program At each local installation, there is a configuration of peripheral devices that are specifically assembled for its needs. Since these peripheral devices share common computer resources, it is extremely important to establish an orderly traffic control. The result is the Queue Manager in the Batch Software System as described in Section 9.2. DEC System-10 provides a service program called CDRSTK to provide these functions. At the University of Pittsburgh, the CDRSTK program has been modified and enriched, and the result is called the Operation Stack (OPRSTK) program. Description of this program and its use has been covered in Chapter 7. For the purpose of completeness, some pertinent portions of its details will again be given here.

To submit a Batch job at a terminal, the following steps should be taken:

---

*Disabled at the University of Pittsburgh and replaced by the OPRSTK program

A. Sign on at a terminal.

B. Create a control file using an editor. Save the file and name it with an extenison of CTL.

C. Run the OPRSTK program by either of the following two monitor command forms:

```
.R OPRSTK
ENTER FILE SPECIFICATION > NAME.CTL
```

or simply,

```
.OP NAME
```

If the control file has an extension of CTL, the extension part may be omitted in the command format. The System will respond with the job card identification and assign a sequence number for identification.

Example: We will now repeat the program used several times in this book as illustration, a problem to solve for a real root of a cubic equation by Newton-Raphson method: $Ax^3 + bx^2 + Cx + D = 0$ with an initial trial value x=X1. Program in FORTRAN is written and stored as NEWTON.FOR. The program listing is shown below:

NEWTON.FOR

```
      READ(5,10)A,B,C,D,X1
   10 FORMAT(5F)
    1 X2=X1-((A*X1**3+B*X1**2+C*X1+D)/(3.*A*X1**2+2.*B*X1+C)
      IF(ABS((X1-X2)/X2)-0.001)3,3,2
    2 X1=X2
      GO TO 1
    3 WRITE(6,11)X2
   11 FORMAT(/' THE REAL ROOT =', F20.7)
      STOP
      END
```

In the example shown below, we will attempt to solve the equation:

$$x^3 - 16x^2 + 65x - 50 = 0 \quad \text{with X1=16}$$

Thus, the input data are 1, -16, 65, -50 and 16 respectively for A,B,C,D,X1. The control file made up is FORT.CTL::

FORT.CTL:

```
$JOB[115103,320571]
$DATA
  1.0  -16.0  65.0  -50.0  16.0
$EOD
.EXECUTE NEWTON.FOR
$EOJ
```

Note that the $PASSWORD card is omitted in the control file because it is unnecessary when the batch jobs are submitted from a terminal.

Once this preparatory work is done, submitting a Batch job is simply issuing a monitor command of OPRSTK (abbreviated as "OP") at the user's terminal:

```
.
$JOB[115103,320571]
;;; END OF JOB AFTER 6 CARDS / SEQUENCE NUMBER IS 3678
```

The output is just a one-liner:"THE REAL ROOT = 10.0". The log file showing the job time history is attached here with comments and annotations.

REFERENCES

1. BEGINNERS GUIDE TO MULTIPROGRAM BATCH (DEC number DEC-10-OMPBA-C-D), third edition, Digital Equipment Corporation, Maynard, Massachusetts; December, 1974.

2. DEC SYSTEM-10 OPERATING SYSTEM COMMANDS (DEC number DEC-10-OSCMA-A-D), Chapter 3: "Batch System Commands", pp. 3.1-3.54, Digital Equipment Corporation, Maynard, Massachusetts; 1974.

3. OPRSTK, DEC-10 NOTES, Computer Center, University of Pittsburgh, Pittsburgh, Pennsylvania; May, 1980.

```
16:39:27 BAJOB  BATCON version 13(1071)-2 running JOBAAA  seq:6839  user:*SZE

16:39:27 BASUM  Time:00:00:30  Core:12K  Unique:YES  Restart:YES  Priority:2

16:39:27 MONTR
16:39:27 MONTR  .LOGIN 115103,320571/LOCATE:6
16:39:28 USER   JOB 5 PITT DEC-1099/B 63A.45B TTY357  Mon 28-Jul-80 1639
16:39:28 USER   [LGNJSP Other jobs same PPN]
16:39:28 USER   Last login: 28-Jul-80 1629
16:39:28 USER   Usage ratio: 0.00   Units used: 19.0
16:39:28 USER   SYS B DOWN 0000-0800 MON AUG 04 FOR REGULAR HARDWARE MAINTENANCE
16:39:28 USER   SYS B DOWN 0000-0300 TUE JUL 29 FOR  REGULAR SOFTWARE MAINTENANCE
16:39:28 USER
16:39:28 USER   DUE TO HARDWARE PROBLEMS THERE IS ONLY ONE DISK DRIVE AVAILABLE
16:39:28 USER   FOR PRIVATE USER PACKS.
16:39:28 USER
16:39:28 USER   ************************************************************
16:39:28 USER   *       PLEASE READ THE FOLLOWING INFORMATION:             *
16:39:28 USER   * FILE                  SUBJECT                            *
16:39:28 USER   *  SYS: NEWS             No fee for 1022 usage             *
16:39:28 USER   *  NEW: TEKLIB.HLP       New Tektronix software           *
16:39:28 USER   ************************************************************
16:39:28 MONTR
16:39:28 MONTR  .
                $JOB[115103,320571]/OUTPUT:5/LOC:6
                $DATA
16:39:28 MONTR  .SET CDR QAA     ;created by OPRSTK
16:39:29 MONTR
16:39:29 MONTR  .
                $EOD
16:39:29 MONTR  .EXECUTE NEWTON.FOR
16:39:30 USER   LINK:   Loading
16:39:30 USER   [LNKXCT NEWTON execution]
16:39:30 USER
                16:39:30 USER   STOP
16:39:30 USER
16:39:30 USER   End of execution    FOROTS 5B(1001)
16:39:30 USER   CPU time: 0.13  Elapsed time: 0.65
16:39:30 MONTR  EXIT
16:39:30 MONTR
16:39:30 MONTR  .
                $EOJ
16:39:30 BLABL  %FIN:
16:39:30 MONTR  .DEL SPL:QAA.CDR      ;created by OPRSTK
16:39:31 USER   Files deleted:
16:39:31 USER   QAA.CDR
16:39:31 USER   18 Blocks freed
16:39:31 MONTR
16:39:31 MONTR  .
                ;;; END OF JOB AFTER 6 CARDS ;;;
16:39:31 MONTR  .KJOB SPL:JOBAAA.LOG=/W/B/Z:5/VR:2/VS:6839/VL:15/VC:0/VT:0/VD:R/VJ:
16:39:32 USER   Total of 1 block in 1 file in LPTS6 request / Sequence number 6839
16:39:32 USER   Other jobs same PPN
16:39:33 MONTR  Job 5 [115103,320571] off TTY357 at 1639 28-Jul-80  Connect=1 Min
16:39:33 MONTR  Disk R+W=66+28  Tape IO=0  Saved all files (18 blocks)
16:39:33 MONTR  CPU 0:00  Core HWM=13P  Units=0.0079 ($0.59)
```

Batch logs in the job.

First line in the control file

Store input data

Monitor command generated by $DATA

Execution

Sign-off the batch job.

monitor commands generated by $EOJ

LOG File for the JOB FORT

CHAPTER 10

TAPE HANDLING

10.1 Magnetic Tape

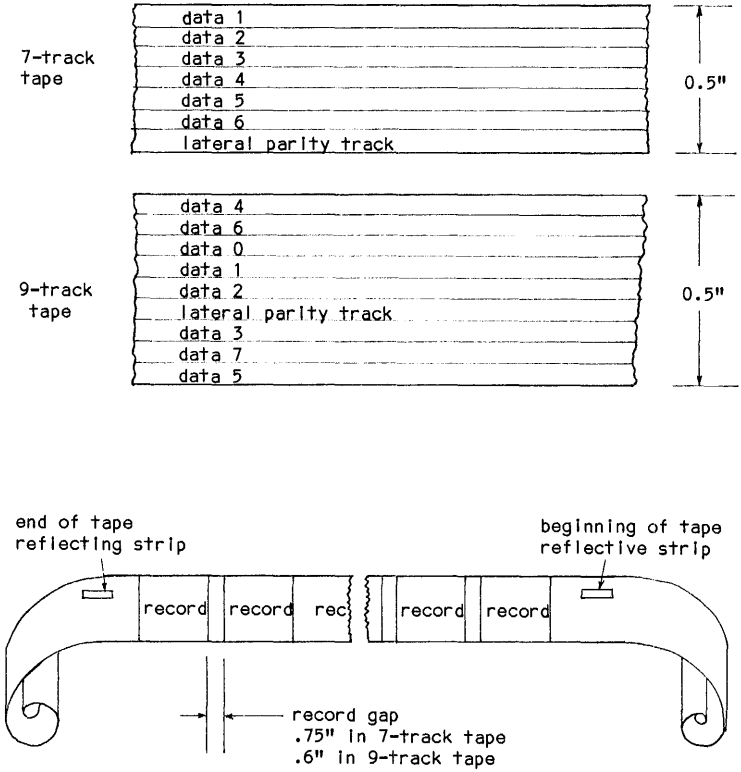
The simplest and the least expensive medium for mass storage is the magnetic tape. It is a plastic tape coated with iron oxide which may be magnetized to record information. A typical 10.5-inch reel of magnetic tape is one-half inch wide and up to 2400 feet long. The width of the tape is divided into either 7 or 9 tracks, one of which is reserved for parity bit track to detect transmission error. Hence, a 7-track tape has six information tracks and a 9-track tape has eight. Characters are written across the width of the tape. Figure 10.1 shows the present industry standard of the magnetic tape format.

To read from or to write onto a magnetic tape, the tape reel is first mounted on a tape transport, also called a tape drive. There is a take-up reel, and a capstan pulling the tape past the read/write heads. Information is written on the tape by sending current through the write-heads to magnetize the tracks; information is read from the tape by sensing the induced voltage as the read-heads pass over the recorded (and magnetized) tracks.

Access speed is essentially a function of the tape speed and the packing density of information. Tape speeds of commercially available tape drives range from 10 to 200 IPS (inches per second). The standard packing densities of a 7-track tape are 200, 556 and 800 BPI (bits per inch), and those of a 9-track tape are 800, 1600 and 6250 BPI. High tape speed coupling with high packing density result in a requirement for higher-speed synchronization, both mechanically and electronically, and such tape drives would be more costly. Typical recording and access rates are 30 to 320 kilo-characters per second.

Magnetic tapes are not inherently designed for a specific packing density, but is usually certified by their manufacturers at a particular BPI. The certification implies that the manufacturer has recorded test patterns at that density and successfully read back the data at or below an allowable error rate. Seven-track and nine-track recordings can be made on the same type of half-inch wide tapes. However, tapes recorded on a 7-track tape drive cannot be played back on a 9-track trape drive, and vice versa. On the other hand, once a tape is erased, it may be used again on either type of drive.

Generally, there are two techniques for synchronization of writing characters on a tape. One requires that the tape has a pre-recorded timing track, which is a track containing all 1's to indicate each character position in the tape. The other is to use an internal clock generator during the write



Records have variable lengths. The industry standard format has 7-track tape records containing from 24 to 4008 characters, and 9-track tape records from 18 to 2048 characters.

Figure 10.1 Magtape Format

operation. When a clock generator is used, it is nearly impossible to guarantee that the tape moves at exactly the same speed between two successive operations, and hence inserting a modified record between two existing records is normally not possible. This necessitates strictly a sequential use of the tape, and modifying a tape really means copying the material with changes onto another part of the tape or onto another tape.

As shown in Figure 10.1, at the beginning of each reel of tape, there is a sensing strip to denote the start of information; at the end there is another strip signaling the end of tape, which prevents the drive mechanism from pulling the tape off its reel. These are respectively the beginning and the end of tape marks (BOT/EOT). Files written on a tape are separated by an end-of-file

mark (EOF) which may be written or sensed by program. In addition, a gap about .75" (for 7-track tapes) or .60" (for 9- track tapes) is inserted between two successive records, not only to delimit two records, but also to allow time for the tape to accelerate or decelerate before the heads reach the beginning of a record.

Many operating system requires that the files be delimited by identifying records, called header labels and trailer labels. These labels are in addition to the end-of-file marks. A typical header label would contain the name (or number) of the file and certain physical characteristic of the file. Programs can be designed to check and verify that the correct tape and correct file have been mounted on the tape unit for use.

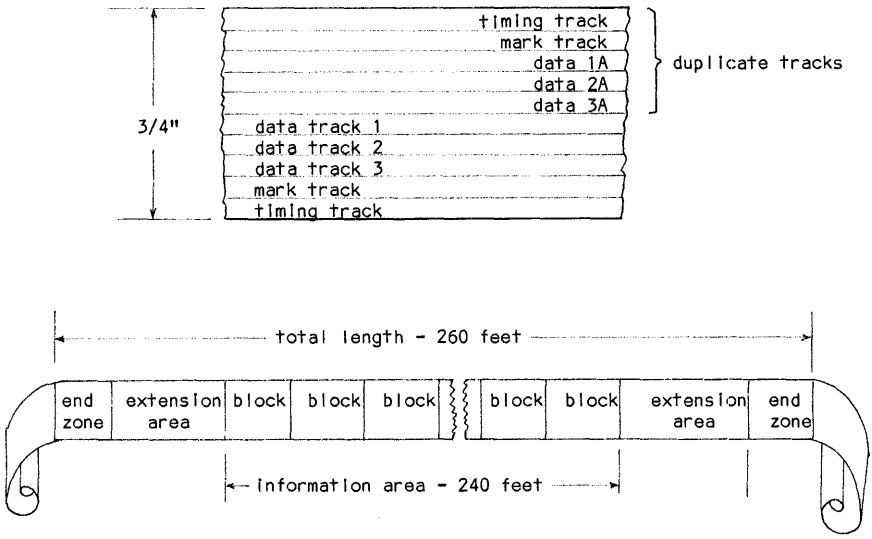
10.2 DECtape ®

During the early sixties, one nonstandard tape system was developed for small computer systems by the Massachusetts Institute of Technology. Digital Equipment Corporation adapted this system and call it DECtape. It is a highly successful magnetic tape and is used extensively on all DEC computers as well as in many minicomputers. It provides a low-cost and highly reliable auxiliary memory. DECtape utilizes a 10-track read/write head. Reliability of storage is accomplished by redundant recording of all data. As shown in Figure 10.2, the track format shows two identical sets of tracks. Redundant recording of each characters bit on non-adjacent tracks greatly reduces bit dropouts and minimizes the effect of skew. The timing and mark tracks are pre-recorded, and they control the timing of operations within the control unit and establish the format of the data contained on the information tracks. Since the tape drive operations and many control function lock-step with the timing-track signals, wide variations of tape speed do not affect its performance.

The data tracks of a DECtape are located in the middle of the tape, where the effect of skew is minimum. The data is one bit position of each track is called a line or a character. Since twelve lines make a word, the tape can record 36-bit data words. During normal data writing, the system disassembles a 36-bit word and distributes the bits so that they are recorded as twelve 3-bit characters.

A 260-foot reel of DECtape is divided into three major areas: end zones, extension zones, and the information zones. The two end zones, each approximately 10 feet, mark the physical ends of the tape and are used for winding the tape around the heads and onto the take-up reel. These zones never contain data. The extension areas mark the end of the information region of the tape. Their length is sufficient to ensure that once the end zone is entered and tape motion is reversed, there is adequate distance for the drive to come up to a proper tape speed before entering the information area. The information area consists of blocks of data. The standard length is 578 blocks, each containing 128 data words nominally. In the DEC System-10 applications, part of the information area is used as file directory, and total usable information area is 574 blocks. The file directory can accommodate a maximum of 22 filenames. Therefore, the capacity of a DECtape used in DEC System-10 is either 22 files or 574 blocks, whichever limit is reached first. Since each block format is symmetric with the block number at both ends, search of blocks may be done in either direction. the DECtape serves not only as a data storage medium, but also as a random access device.

®Registered trademark, Digital Equipment Corporation, Maynard, Mass.



DECtapes used for DEC System-10 have a capacity of 574 information blocks, and its file directory has a capacity of 22 filenames. A DECtape reaches the limit of its capacity when either of these two limits is reached.

Figure 10.2 DECtape Format

In comparison, DECtape has higher-performance, and is more reliable and convenient to use than the magnetic tape. Being a directory device, it may be used in the same way as the disk, but with more storage capacity for individual users. On the other hand, it storage capacity is puny by comparison: A 2400'-reel magtape at 6250 BPI can store about 125 millions characters, while a DECtape reel can only store about 300,000 characters.

10.3 Preliminary Procedures

After a user acquires reels of magtape (coined word for magnetic tape) or DECtape, there are certain preliminary procedures that should be performed. A typical Computer Center installation stores at its premise hundreds or thousands of reels of tapes. Among the magtapes, some are 7-track tapes, and some are 9-tracks. Even for the same type of magtapes, they may have different packing density. Although the system installation can handle all various combinations, it will be extremely important to reduce the chance of human error of mounting a wrong tape, however rare the chance may be.

Each computer installation will generally develop a security verfication procedure for tape mounting. These procedures vary from one installation to another. The procedure described below is implemented at the University of Pittsburgh. Users at other installations should inquire at their Computer Center.

Two preliminary steps are involved:

(1) Registration of tapes This is a clerical step of registration of tapes with the personnel of the Computer Center. A Pitt VID (Visual Identification) is issued upon registration. The VID is a serial number with either an A-prefix or a B-prefix. They are respectively for DECtapes and magtapes. For example, A1004 and B313 are the VIDs of a DECtape and a magtape respectively. Once a VID number is assigned, it is displayed outside of the tape reel. The VID is an essential part of identification that must be given in a MOUNT command.

(2) Labeling of tapes For DECtapes and 9-track magtapes, an additional means of safeguarding the identification of tapes is available. The numerical part of the VID is recorded in the first file of the tape. When the tape is mounted, not only the operator will seek the right reel with the specified VID, the system will read the first file to get the recorded VID and verify that against the VID given by the user in his MOUNT command. The labeling process does not apply to 7-track tapes. The process of labeling tapes is accomplished by calling a program named TAPLBL, after the tape has been mounted. Call for the TAPLBL is done by a monitor command:

```
R TAPLBL
```

When a prompt symbol returns, apply a labeling command of the format:

```
DVNAME:  /density switch
```

where DVNAME: = physical or logical name of the tape drive;

/density switch = a switch to specify packing density.

Acceptable density switches are:

```
/8 = 800 BPI for MT8:
/1 = 1600 BPI for MT8: or MT9:
/6 = 6250 BPI for MT9: only
```

The density switch is not applicable to the DECtapes or 7-track magtapes. One caution should be exercised in using the TAPLBL program: It will write the label onto the first file on the tape. If the tape is a DECtape or a blank magtape, it does not matter. If the tape contains some stored information already, using the TAPLBL would destroy the first file. Therefore, in labeling a magtape with stored information, a scratch tape should be used to copy the files. After the labeling process, the contents of the scratch tape may be copied back. These are now illustrated by three examples below:

Example: To label a DECtape whose VID is A1004 which is already mounted on the DECtape drive with a logical name of T1:

```
.DRIVES DTA
.MOUNT DTA:T1/WE/VID:A1004/NL
.R TAPLBL
*T1:
.DISMOUNT
.UNDRIVE
```

Example: To label a 9-track blank magtape for 800 BPI. Assume the VID to be B313 and the logical name of the tape drive to be T1:

```
.DRIVES MT8
.MOUNT MT8:T1/WE/VID:B313/NL
.R TAPLBL
*T1:/8
.UNDRIVE          (UNDRIVE will force a DISMOUNT.)
```

Example Magtape B123 has prestored information, and magtape B124 is a scratch tape. The following shows first to "park" the content of B123 in B124. After the label is made, the information is copied back:

```
.DRIVES MT9(2)
.MOUNT MT9:T1/WE/VID:B123/NL
.MOUNT MT9:T2/WE/VID:B124/NL
.R MTCOPY
*T2:=T1:                Park infor. of T1 in T2 temporarily
.R TAPLBL
*T1:                    Make label on T1
.R MTCOPY
*T1:=T2:                Copy back info from T2 to T1.
.DISMOUNT T1:
.DISMOUNT T2:
.UNDRIVES
```

With a tape thus registered and labeled, the user is now ready to process his tape.

10.4 Allocation of Tape Drives and Mounting of Tapes

The reservation for tape drives and the mounting of a user's tape precedes the actual tape processing. Tape drives are restricted devices and are made available to user only at a reservation request (by the command DRIVES). Mounting of a tape requires a physical effort from the machine operator, since these tapes are stored off-line at the Computer Center premise. A monitor command MOUNT, along with its associated switches, will tell the operator to mount the user's tape. Details of the two important commands DRIVES and MOUNT, and their companion commands UNDRIVES and DISMOUNT, have been covered in Chapter 8 (Sections 8.8,8.9,8.10)

10.5 Sequential Processing of Magtapes

The DECtape is a directory device, in which files are represented by the standard file specification of DEV:NAME.EXT. The users need not be concerned with the physical location of the read/write heads with respect to the tape track at any time. From the user usage point of view, a file on a DECtape is no different from one on disk. There is a difference in access time, of course. But as far as commands and instructions are concerned, what can be applied to a disk file can also be applied to a DECtape file, that is, once the DECtape is mounted.

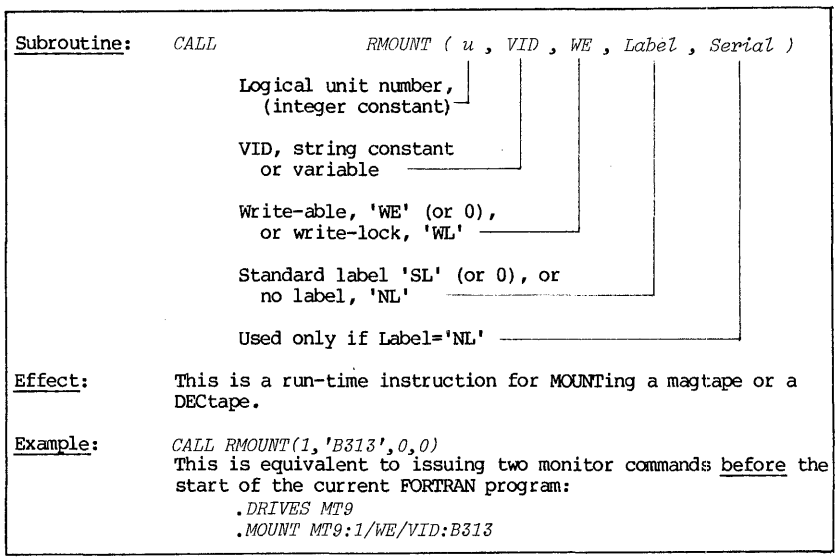
The magtape unit, however, is quite different. There is no directory or filenames. Records are separated from each other by the end-of-record mark, and the files are separated from each other by the end-of-file mark. The only identity a tape file has on a tape is that it occupies a certain sequential file position, such as file No. 4. Therefore, the principal way of locating some information on a magtape is to start at some reference point (such as at the beginning of a tape) then to go forward (or sometimes go backward) certain number of records or certain number of files. Hence, processing of tapes deals a great deal with the sequential positioning of the tape. The following are several PIP commands with magtape switches and their monitor command equivalents:

PIP Command	Function	Equivalent Monitor Command
*MT9:(M#NA) =	Advance the tape N files	.SKIP MT9: N FILES
*MT9:(M#ND) =	Advance the tape N records	.SKIP MT9: N RECORDS
*MT9:(M#NT) =	Advance to the end of tape	.SKIP MT9: EOT
*MT9:(M#NB) =	Backspace N files	.BACKSPACE MT9: N FILES
*MT9:(M#NP) =	Backspace N records	.BACKSPACE MT9: N RECORDS
*MT9:(MF) =	Mark end-of-file	.EOF MT9:
*DEV:(MW) =	Rewind the device, where DEV = DTA,MT7,MT8,or MT9	.REWIND DEV:

If the command indicates "MT9:", that command also applies to MT7: and MT8: In a typical application, a magtape is used only as a means of mass storage. Data on tape will normally be transferred to the disk first for processing, and results will then be copied back to the tape.

10.6 FORTRAN-10 Execution-Time Tape Control

The foregoing discussions dealt with sequential tape processing using a processor such as PIP to handle file management tasks. If the tapes are used as the data files for input/output, there must be commands in the language processor to perform these tape handling tasks at execution time. FORTRAN-10 has a subroutine RMOUNT (developed at Pitt) to mount the tape and a group of tape handling FORTRAN statements. They are outlined below:



Once the tapes are properly mounted, the FORTRAN-10 tape control statements may be applied to control these devices.

The device control statements are now summarized below which has been presented before in Chapter 3 as Table 3.22.

Statement	Function
REWIND u	Move and re-position the file back to the first record.
UNLOAD u	Rewind the source reel so that the tape is complely off the take-up reel. The tape will be ready for unloading.
BACKSPACE u	Backspace one record except if it is already at record No.1. This statement cannot be used for files set up for random access,list-directed,or NAMELIST-controlled I/O operations.
ENDFILE u	Write an endfile record in the file located on device u.
SKIP RECORD u	Skip one record on device u.
SKIP FILE u	Skip one file which follows immediately the current one.
BACKFILE u	Backspace to the first record of the file preceding the current one.

In all above statements, "u" = specified logical unit number.

TAPE SERVICE PROGRAMS

Several tape service programs will be included here in summarized forms. In the cases where a magtape or a DECtape is involved, it will be assumed that the proper preliminaries of getting a tape drive, mounting the tape and assigning a logical name have already been done. In the examples, we will use the VID of A1003, A1004, ..., B313, B314, etc., as the tape registry numbers. The logical names used will be T1:, T2:, etc., except in the case of the UARC program.

10.7 The UARC Program

The UARC (User ARChive) program is a service program for maintaining disk files by copying them onto a user's UARC tape for safekeeping. When the user's magtape is being MOUNTed, it is required that its logical name be given as UARC. Thus the preliminaries of using this program should be two monitor commands as given below:

```
.DRIVES MT9
.MOUNT MT9:UARC/WE/VID:B313
```

The UARC program may be called by the monitor command:

```
.R UARC
(message)
*
```

After a brief message, the terminal types out a prompting "*" and UARC is ready to accept commands. Use CTRL-Z to exit.

Example: **DIRECTORY *.FOR*
Function: List on the terminal all FORTRAN files on the UARC tape.

Example: **DIRECTORY LPT: = *.FOR*
Function: List on the line printer all stored FORTRAN files on the UARC tape.

Example: **DIRECTORY UARC.DIR = *.**
Function: Prepare a disk file UARC.DIR that has a directory of the UARC tape.

Example:
```
*FREEZE *.FOR
*THAW NEWTON.FOR
```
Function: Copy all FORTRAN disk files onto the tape. If the tape already stored certain FORTRAN files with the same names, copying will not be done if the disk files are older (in creation dates). The THAW command recalls to the disk a tape file named NEWTON.FOR.

Example: The following sequence shows how to "clean up" a UARC tape by erasing all older versions of the same program. Two tape drives and two tapes are required.

Items	Explanations and Examples
Program Name	UARC
Calling Sequence	*.R UARC* *command/switches
Functions	(1) To store backup copies of disk files on user's UARC tape. (2) To recall UARC back copies of files back to the disk. (3) To report the content of the UARC tape.
Commands	**CLEAR SERNO = Bxxxxx* To clear the tape of all previous files and write a tape label of "UARC" in the first file. This is required preliminary of tape preparation for UARC application. *Bxxxxx* = tape serial number.
	* *DIRECTORY output file spec = input file list* Similar to the monitor DIRECT. Wild card construction is permitted for the input file list.
	* *FREEZE list* To copy the disk files in the "list" onto the UARC tape without deleting them. If there is already a file of the same name on the UARC tape, copying will be done only if the creation date of the disk file is more recent.
	* *THAW list* To restore files from the tape to the disk. If there is already a file on disk with the same name, restoration will be done only if the creation date of the tape file is more recent.
	* *UPDATE* To store on the UARC tape all new and newer versions of disk files.
	* *COPY SERNO = Bxxxxx* When a second tape is MOUNTed and is named as UCOPY:, this command will make it a UARC tape and further commands copy all current files onto it. This is the only way a UARC tape may be cleaned up and old useless files amy be erased. See example below.
	* *SAVE TAPE.DIR* * *RESTORE TAPE.DIR* To save the current UARC directory on disk for thawing.
Switches	* *DIRECTORY/ALL* Include all old versions in the report. * *FREEZE/CHECK* *THAW/CHECK* *UPDATE/CHECK* Report the files that would be transferred. * *FREEZE/VERIFY* *UPDATE/VERIFY* *COPY/VERIFY* Report successful transfers

Table 10.1 Summary of the UARC Program

```
.DRIVE  MT9(2)
.MOUNT  MT9:UARC/WE/VID:B313
.MOUNT MT9:UCOPY/WE/VID:B314
.R UARC
*COPY SERNO = B314
*↑Z
.R MTCOPY
*UARC: = UCOPY:
*↑Z
.DISMOUNT UARC:
.DISMOUNT UCOPY:
.UNDRIVE
```

10.8 The ACCESS Program

The ACCESS program is a tape maintenance program developed at the University of Pittsburgh. It allows the owner of tapes to specify and modify the access protection, to add or delete PPNs for authorized access, to attach comments to any tape, to request removal of tape from the library either temporarily or permanently, and to report the status and directory of tapes.

Calling and execution of this program does not actually involve physically with any tape. Therefore, no preliminaries of reserving a tape drive or mounting a tape is required.

The calling sequence for the program is as follows:

```
.R ACCESS
(message)
*command/switches
```

Use CTRL-Z to exit from the program.

The commands and the switches of the ACCESS program are summarized in Table 10.2. Note the notations used for PPN:

```
   PPN = owner's PPN under which the tape is registered.
AUXPPN = PPN authorized by the owner to have access to the tape.
```

The tape directory has wide print format. If the directory will be listed on the terminal, a monitor command of "TTY WIDTH 132" should first be applied.

Example: *DIRECTORY*
Function: List on the terminal all tape status registered to the PPN.

Example: *DIRECTORY/ON:LPT:*
Function: Print the status report on a line printer.

Example: *TAPE/ADD/AUXPPN:[123456,654321]/PROTECT:PPPP/VID:A1003*
Function: Add an auxiliary PPN [123456,654321] to have access to the DECtape A1003 with an access authorization of PPPP.

Items	Explanations
Program Name	ARCHIVE
Calling Sequence	.*R ARCHIVE* *command/switch
Functions	(1) To store a disk file on the ARCHIVE tape and remove the disk file from the disk. (2) To restore a frozen file on the ARCHIVE tape back to the disk. (3) To report the directory of the frozen files.
Commands	**DIRECTORY output = input list* To list the directory of frozen files. Similar format as the monitor command DIRECT.
	**FREEZE list* To move the disk files in the "list" to the ARCHIVE system.
	**THAW list* To restore as disk files those in the "list".
	**DELETE list* To delete the listed files from the ARCHIVE tape.
	**PROTECT file<xyz>* To change protection code of a frozen file to < xyz>.
Switches	/LIST Used with *THAW*, *FREEZE*, *PROTECT* and *DELETE* to list the pending request.
	/KILL Used with *THAW*, *FREEZE*, *PROTECT* and *DELETE* to delete the pending requests.

Table 10.3 A Summary of ARCHIVE Commands and Switches

10.10 The CHANGE Program

The CHANGE program is a tape-translation program that converts the files on a magtape of one format to an output (usually a disk) of another format. The program is called by a monitor command:

```
RUN  PRG:CHANGE
```

After the CHANGE program is called, command may be applied which has the format of:

```
Output spec/output switches = Input spec/input switches
```

Switch	Argument	Explanation
/BUFFERS	:n	n = number of buffers to be set up
/ADVANCE /BACKSPACE	:n :n	Advance or backspace n files before processing.
/BLOCK	:n	n = blocking factor in number of records per block
/RECORD	:n	n = number of characters per record
/DENSITY	:BPI	BPI = tape density in BPI: 556,800,1600,or 6250
/LABEL	:arg	The argument "arg" is one of the following: none, DIGITAL, BURROUGHS, IBM, or GE635.
/MODE	:arg	The argument "arg" is one of the following: ASCII 7-bit ASCII code HPASCII 8-bit ASCII IMAGE 36-bit DEC-10 word EBCDIC EBCDIC code
/PARITY	:arg	The argument "arg" is either ODD or EVEN.
/INDUSTRY /NOINDUSTRY		Initialize for industry compatible 9-track tape. Turn off INDUSTRY switch.
/SCAN /NOSCAN		Scan file for file named Turn off SCAN switch.
/ERROR /NOERROR		Report parity and checksum errors. Turn off ERROR switch
/REWIND	:arg	The argument "arg" is one of the following: BEFORE Rewind before processing. AFTER Rewind after processing. ALWAYS Rewind always OMIT Rewind neither before nor after.
/LIST /NOLIST		List the device directory. Turn off the LIST switch.
/FLIST /NOFLIST		List the device directory, file names only. Turn off the FLIST switch.
/CRLF /NOCRLF		ASCII file has CR-LF after every record. Turn off the CRLF switch.

Table 10.4 A Summary of Selected CHANGE Switches

The magtape is the most frequently used low-cost portable bulk storage medium. Unfortunately, there are many different tape formats and tape produced on one type of computer generally cannot be directly run on another type. There are differences in coding method, block sizes, word sizes, tape density, parity system, etc. When a tape containing data or program is obtained from another installation that has a different type of computer, the tape must be first translated to the local format before it may be used. Thus, when the CHANGE program is used for the translation, the input would be the MOUNTed "foreign" tape, and the output would frequently be a disk file. The same set of switches is applicable for both the input and the output. Selected switches are tabulated in Table 10.4. A complete description of the CHANGE program, with commands and switches, is given in Reference 6.

Example:

```
.DRIVE  MT8
.MOUNT  MT8:T1/WL/NL/VID:B313
.RUN PRG:CHANGE
CHANGE  08:44   11/18/80
READY.
> DISK.FOR/MODE:ASCII/RECORD:80/BLOCK:10-
# MT8:/MODE:EBCDIC/RECORD:80/BLOCK:10/INDUSTRY-
# /LABEL:IBM
```

Function: This procedure will convert a foreign tape produced from an IBM/360 or 370 to a ASCII disk file named DISK.FOR. A dash at the end of a CHANGE command line indicates that the command is to be continued on the next line. In response, CHANGE returns a different "#" prompt symbol on the next line.

10.11 Tape Transfer and Comparison Programs - MTCOPY, DTCOPY and FILCOM

Three service programs associated with tape-to-tape transfer and comparison tasks are included here:

MTCOPY: magtape-to-magtape transfer

DTCOPY: DECtape-to-DECtape transfer

FILCOM: Verification of transfer by file comparison

They are summarized in Tables 10.5, 10.6 and 10.7.

Items	Explanations
Program Name	MTCOPY
Calling Sequence	*.R MTCOPY* **outtape:/outswitches = intape:/inswitches/funcswitches* where outtape: = physical/logical name of destination tape. intape: = phyiscal/logical name of source tape. outswitch = switch for output tape characteristics inswitch = switch for input tape characteristics funcswitch = function switch
Function	To copy the contents of an input tape onto an output tape.
Input switches Output switches	Must be placed on the proper side of the command. /6 6250 BPI (9-track tape only) /1 1600 BPI (9-track tape only) /2 200 BPI (7-track tape only) /5 556 BPI (7-track tape only) /8 800 BPI (7 or 9-track tape) /A:N Advance N files before operation /B:N Backspace N files before operation /E even parity /I IBM compatible mode /R Rewind before and after operation (default) /U Unload after operation /Z Suppress monitor error recovery on READ errors
Function Switches	May be placed on either side in the command. /C:N Copy N files /C Copy to double end-of-file marks /G Proceed on errors /M:NNNN Set maximum blocksize permitted to NNNN /N Suppress automatic rewind /V:N Verify to double end-of-file marks
Default conditions:	What will happen if no switch is placed on the command: 1. Density is 6250 BPI for 9-track, 800 BPI for 7-track unless there is a SET DENSITY monitor command applied. 2. Odd parity, DEC-compatible, maximum blocksize=1024. 3. Copy and verify tape. Rewind before copying and verifying. Stop on errors.
Examples:	**TP1: = TP2:/8* Copy a tape at 800 bpi. New tape is on TP1: **NEW:/R/A:3 = OLD:/R/A:2/C:3/V* Copy files No.3,4,5 on OLD: onto the NEW: tape as files No.4,5,6.

Table 10.5 A Summary of MTCOPY Program

Items	Explanations
Program Name	DTCOPY
Calling Sequence	*.R DTCOPY* **OUTPUT: = INPUT:/switches* where OUTPUT: = physical/logical name of the destination DTA. INPUT: = physical/logical name of the source DTA
Function	1. To copy the entire content of one DECtape onto another. 2. To clear the output DECtape. 3. To compare word by word two DECtapes. 4. To load a bootstrap loader.
Switches	/C Copy all blocks. /N No directory listing. /V Comapre two DECtapes (verifying) word by word. /Z Zero out the output DECtape.

Table 10.6 A Summary of DTCOPY Program

Items	Explanations
Program Name	FILCOM
Calling Sequence	*.R FILCOM* **output = input1, input2/switches* where OUTPUT = output file specs; default is TTY: input1, input2 = two input file specifications
Function	To comapre two versions of a file and output the differences.
Switches	/Q Print out the message "?FILE ARE DIFFERENT," but do not list the differences. /A Compare two files in ASCII codes. /B Do not disregard blank lines. /S Ignore spaces and tabs in the comparisons.

Table 10.7 A Summary of FILCOM Program

REFERENCES

1. OPERATING SYSTEM COMMAND MANUAL, DEC-10-OSCMA-A-D, Digital Equipment Corporation, Maynard, Massachusetts; 1974

2. TAPE UTILITY PROGRAMS, the Computer Center, University of Pittsburgh, Pittsburgh, Pennsylvania; April, 1980.

3. TAPE PURCHASE, REGISTRATION AND REMOVAL, the Computer Center, University of Pittsburgh, Pittsburgh, Pennsylvania; March, 1977.

4. PDP11 PERIPHERAL HANDBOOK, Digital Equipment Corporation, Maynard, Massachusetts; 1975

5. UTILITY MANUAL, DEC-10-UTILA-A-D, Digital Equipment Corporation, Maynard, Massachusetts; 1975

6. System HELP files and Program Library HELP-files: SYS:ACCESS.HLP (May 19,1980), SYS:MTCOPY.HLP (August 20,1979), SYS:FILCOM.HLP (August 22,1978), PRG:CHANGE.HLP (October 24,1980), University of Pittsburgh, Pittsburgh, Pennsylvania.

APPENDIX A

A SUMMARY OF PIL LANGUAGE

PIL ("Pittsburgh Interpretive Language) is a conversational language which contains extensive man-machine interactive facilities to provide assistance in error diagnosis and error recoveries. It is much more error-tolerant than the conventional algebraic languages such as FORTRAN. PIL was included in the First and the Second Editions of this book (two chapters). With the rise in computational maturity of the user community, even for the new users, PIL becomes less important than before. Therefore, only a summary will be included in the Third Edition of the book. Users interested in more details should consult either Reference 5 or Reference 6.

The PIL processor may be called by a monitor command:

```
R PIL
```

The computer will respond by typing out at the terminal:

```
READY:
*
```

The PIL processor is now ready to process the user's program or commands.

A.1 Rules on PIL Variables, Constants and Expressions

(1) PIL variables:

a. Variable name must begin with a letter, and must not be longer than ten characters.

b. Upper and lower cases are different variables. (FORTRAN users, beware)

c. All numerical variables are real variables.

(2) Constants

a. There are 3 types of PIL constants, numeric, string and Boolean.

b. Numeric constants are real constants, ranging from E-42 to E+34 with 8 decimal digit precision.

c. String constants are enclosed in quotes, such as "TIME-SHARING"

d. Two values for Boolean: THE TRUE and THE FALSE.

(3) Subscripted variables:

a. There is no practical limit of dimension of subscripts.

b. Subscripts may be positive, zero, or negative.

c. No declaration statement is needed for dimension.

(4) Expressions:

a. Arithmetic expressions: PIL arithmetic expressions follow the same rules as those for the FORTRAN language. The PIL library functions are tabulated in Table A.1.

b. Boolean expression: Expressions that are really true-false questions. Boolean operatiors are tabulated in Table A.2.

c. String expressions: String variables may have the following operations:

Concatenation:	For example, "ABC"+"xyz"="ABCxyz"
Masking:	By use of SUBS function.
String multiplication:	Same as multiple concatenation.

Table A.3 shows a list of PIL string functions.

A.2 Statement Labels

In a stored PIL program, each statement carries a numerical label for identification. The label has a format of "mmmm.nnnn" where mmmm is a four digit part number and "nnnn" is another four-digit step number. Thus the label takes on a form of a decimal number, and as in conventional practice, the leading and the trailing zeros are omitted. For example, the label of 1.25 means Part 1, step 2500.

A.3 Some Basic PIL Statements

Substitution:
```
SET X=E
SET  X1=E1,X2=E2,X3=E3,...
```
where X=variable, and E's=expressions.

Transfer:
```
TO step m.n
TO part m
```
where m.n is the statement label.

Execution:
```
DO step m.n
DO part m
```

Termination:
```
DONE
STOP
EXIT
LOGOUT
```

Subprogram Format	Meaning	Examples	
		Format	Value
ABS OF x, \x\	Absolute value of x	ABS OF -3.12, \-3.12\	3.12
SQRT OF x	$\sqrt{x}$	SQRT OF 2.1	1.4491376
SIN OF x	sin x	SIN OF 0.32	0.31456656
COS OF x	cos x	COS OF 0.32	0.95923542
SIN OF x/COS OF x	tan x	SIN OF 0.32/COS OF 0.32	0.33138940
ATAN OF x	$\tan^{-1}$ x	ATAN OF 1.5	0.98279374
LOG OF x	$\log_{10}$ x	LOG OF 2	0.30102998
LN OF x	$\log_e$ x	LN OF 2	0.69314718
ANTILOG OF x	10^x or antilog_{10} x	ANTILOG OF 2	100.0
EXP OF x	e^x	EXP OF (-2)	0.13533528
IP OF x	Integer part of a number x	IP OF 3.21	3.0
FP OF x	Fraction part of a number x	FP OF 3.21	0.21
XP OF x	Exponent of x	XP OF 12.34	1.0
DP OF x	Digit of x	DP OF 12.34	1.234
MAX OF (a,b,c,...)	Maximum value of a set of numbers a,b,c,...	MAX OF (1,2,3)	3.0
MIN OF (a,b,c,...)	Minimum value of a set of	MIN OF (1,2,3)	1.0
RN OF x	Assign a random number between 0 and 1 to the variable x	RN OF x	x=0.65302564

The argument of the subprogram RN OF must be a variable. The argument of any other above subprogram may be a constant, a variable or an arithmetic expression.

Table A.1 PIL Arithmetic Library Functions

PIL Operators		Meaning	Examples
short Form	Long Form		
<	$LT	$<$	A<B
=< or <=	$LE	$\leq$	A $LE B
=	$EQ	$=$	A=B
	$NE	$\neq$	A $NE B
=> or >=	$GE	$\geq$	A $GE B
>	$GT	$>$	A>B
&	$AND	$\cap$	X>3 $AND X<10
	$OR	$\cup$	X<3 $OR X>10
	$NOT	$\overline{X}$	$NOT X>3
	$XOR	$\oplus$	A $XOR B

Table A.2 PIL Boolean Operators

String Functions		Meaning	Examples
short Form	Long Form		
L OF X	LENGTH OF X	length of a string	L OF "ABCDEF" = 6.0
UPPER OF X	UPPER CASE OF X	Force all letters in X to upper cases.	UPPER OF "abcde" = "ABCDE"
LOWER OF X	LOWER CASE OF X	Force all letters in X to lower cases.	LOWER OF "ABCDE" = "abcde"
n $FC X	THE FIRST n CHARACTERS OF X	masking	2 $FC "ABCD" = "AB"
n $LC X	THE LAST n CHARACTERS OF X	masking	2 $LC "ABCD" = "CD"
	THE VALUE OF X	To convert a string of numerical characters to numerical values.	THE VALUE OF "3.1" = 3.1
	THE BCD VALUE OF X	To convert a string of numerical value to a numerical character string.	THE BCD VALUE OF 3.1 = "3.1"
SUBS OF (S,A,L)	SUBSTRING OF (S,A,L)	To mask a string S from its Ath char. for a length of L characters.	SUBS OF ("TIMESHARING",3,4) = "MESH"

Table A.3 PIL String Library Functions

Conditional: m.n IF b, OS

where m.n=statement label of this statement
b=Boolean expression
OS=an object statement

Variations:

m.n IF b, OS1; ELSE OS2
m.n IF b, OS1; OS2

A.4 Loop Statements

(1) Specified indexes: FOR i=m1,m2,...,mn: OS

(2) Unity increment: FOR i=m to n: OS

(3) Specified increment: FOR i=m to n by p: OS

(4) Specified decrement or special terminating condition:

FOR i = m by p UNTIL b: OS

FOR i = m by p WHILE b: OS

A.5 Input/Output Statements

(1) General form:

Demand list

TYPE list

(2) Input with free format:

DEMAND IN FREE FORM, list

where "list" contains the variables in the input list.

(3) Input/output with format

DEMAND IN FORM i, list

TYPE IN FORM i, list

A.6 Input/Output Format

Format specification:

FORM i.
(Specify format field on the second line.)

F-type and I-type: _._ __

E-type: .....

Combined format: __.__!!!!

String field: ########

Field separation and termination: "\" symbol, for example:

-----._\---.----

Variable field length:

%######### %__._

A.7 Subprogram Statements

Subprogram defining statement:

PROCEDURE XXXXXX[X1,X2,...,Xn] = PART m

where: PROCEDURE = a PIL keyword to specify a subprogram

XXXXXX = name of the subprogram, and also the name of the variable whose value is computed by the subprogram, and after execution of subprogram, returned to the main program. All intermediate results are deleted after the execution of subprogram.

If the subprogram is to return multiple value answer, for example,roots of an equation, XXXXXX will then be an array and represented by a subscripted variable.

[X1,X2,...,Xn] = a set of dummy parameters passed to the subprogram. Use square brackets.

PART m = the part specified as the subprogram. Part m must be so written that X1,X2,...Xn are the input parameters and XXXXXX is the computed result.

Subprogram execution statement:

DO PROCEDURE XXXXXX[A1,A2,...,An]

where A1,A2,...,An are the input parameters to be passed to the subprogram.

A.8 File Management Statements

To save a program:

```
SAVE as "FLNAME", list
```

To delete a file:

```
DELETE FILE "FLNAME"
```

To edit a program:

```
CHANGE "OLD" TO "NEW" IN STEP m.n

CHANGE "OLD" TO "NEW" IN FORM i
```

To change a step or a part number:

```
REDEFINE STEP  m.n TO p.q

REDEFINE PART   m  TO  n

REDEFINE FORM   M  TO  n
```

To delete items in the program:

```
DELETE list
```

To load a program:

```
LOAD "FLNAME.EXT[m,n]"
```

where: FLNAME = file name
EXT = extenison of file. Default extension is PIL.
[m,n] = PPN of file owner. Default is user's own PPN.

To attach a file:

```
ASSIGN "FLNAME.EXT[m,n]" AS "FN"

ASSIGN "SCRATCH FILE" AS "FN"

ASSIGN DEV-NAME AS "FN"
```

To delete an assignment:

```
DELETE ASSIGNMENT "FN"

DELETE ALL ASSIGNMENTS
```

A.9 File Input/Output

File Input:

```
READ FROM "FN", list
READ FROM "FN", IN FORM i, list
READ FROM "FN", IN FREE FORM, list
```

File Output:

```
WRITE ONTO "FN", list
WRITE ONTO "FN", IN FORM i, list
```

A.10 File Control Statements

To mark the end-of-file on a file:

```
MARK FILE "FN"
```

To rewind a file:

```
REWIND "FN"
```

To forward or backspace a file:

```
FORWARD SPACE n RECORDS ON "FN"
BACK SPACE n RECORDS ON "FN"
```

To specify end-of-file action:

```
ON FILEMARK "FN", DO PART m
```

To cancel an end-of-file action already specified:

```
ON FILEMARK "FN"
```

To store an assignment:

```
CATALOG "FN" AS "FILNAME.EXT"
```

A.11 Execution-time Function and Program Step Input

To furnish a program step at execution time, such as the step "m.n SET X=A":

```
ENTER "m.n SET X=A"
```

or,

```
DO STRING "SET X=A"
```

A.12 PIL-FORTRAN Linkage

PIL language provides the advantage of conversational mode and free form input format. Error recoveries and error diagnostic provisions further enhance its uses. Consequently, a PIL program is very suitable for the input phase of a program, where the man-machine interaction is at its highest.

Once execution starts, PIL program exhibits an excruciatingly slow speed of execution. It is primarily because PIL is an interpretive language. Every time a PIL statement is to be executed, it must first be translated. Thus in a PIL program, the program is interpreted and executed at the speed of one statement at a time. Error detection for debugging becomes simple because the program will stop at the step where error occurs.

On the other hand, the entire FORTRAN program is compiled at one time, and execution takes place after the compiling. If a FORTRAN program is already compiled, the compiling stage is omitted. Thus a FORTRAN program is generally much faster to run than a PIL program. FORTRAN, however, has its drawback. Man-machine interaction can be implemented only at the expense of core storage for many printout formats, and an interactive program tends to be larger. Also, formats in FORTRAN are more restrictive.

All of these adds to the fact that PIL is superior in interaction but inferior in speed, while FORTRAN is just the opposite. Thus, a compromise is to use a PIL program for the data-input phase, store the data as disk files, and then switch to a FORTRAN program for execution, which will read the stored data file as inputs.

The PIL statements for the PIL-FORTRAN linkage are:

```
EXECUTE "FLNAME"
```

or,

```
RUN "PRGM"
```

where FLNAME = the name of the FORTRAN (with FOR or REL extension) program, and

PRGM = the name of the execution file with EXE extension.

One such application is the Interactive Engineering Program Library. The Programs in the Library are so structured that the PIL phase handles the input of data and problem definition, and the FORTRAN phase handles the execution. The details of the Library are given in Appendix B.

REFERNCES

1. PIL, Class Notes for Engineering Analysis II, T. W. Sze, University of Pittsburgh, Pittsburgh, Pennsylvania; 1968.

2. PIL/L PITT INTERPRETIVE LANGUAGE FOR THE IBM/360 MODEL 50, the Computer Center, University of Pittsburgh, Pittsburgh, Pennsylvania; 1969.

3. A PRIMER FOR PITT TIME-SHARING SYSTEM(PTSS), Chapter 3, T. W. Sze, University of Pittsburgh, Pittsburgh, Pennsylvania; 1970.

4. PIL REFERENCE CARD, the Computer Center, University of Pittsburgh, Pittsburgh, Pennsylvania; 1977.

5. INTRODUCTION TO DEC SYSTEM-10: TIME-HARING AND BATCH, First and Second Editions, Chapters 2 and 3, T. W. Sze, University of Pittsburgh, Pittsburgh, Pennsylvania; 1974 and 1977.

6. PIL, PITT INTERPRETIVE LANGUAGE, Brent J. Ermlick, the Computer Center, University of Pittsburgh, Pittsburgh, Pennsylvania; 1979.

APPENDIX B

INTERACTIVE ENGINEERING PROGRAM LIBRARY

A time-sharing interactive program library has been set up and in operation successfully to serve the faculty and the students in the School of Engineering since 1967. It has been one of the major tools in the computer-aided analysis/design instructions and research in the School. These programs were all designed to operate in the conversational mode, in modular form, and they are application-independent. Thus a differential equation solver program can serve those who are doing work in a wide variety of fields, such as circuit analysis, control system, vibration, structural analysis, process control, etc. In this regard, these programs should also be useful to those in natural, medical and social sciences who require the same type of mathematical techniques but in a different field.

For the Interactive Engineering Program Library, a user with little or no prior computer experience can select and use a program in the Library to solve his problem after a short practice session at a terminal. All instructions of using a program, once its execution begins, will be supplied at the user's terminal during execution. Thus there is no need for the user to study voluminous manual materials in order to use the program.

The structure of each of the Library program utilizes the PIL-FORTRAN linkage technique discussed in Appendix A. Using one of the Library program REALEQ (Simultaneous Linear Equations with Real Coefficients) as an illustration, the linkage works this way:

(1) The user calls for the ENG:REALEQ.PIL program.

(2) Interacitvely the user inputs the data, defines the problem and selects the options, if any.

(3) PIL program then stores these data and options in a disk file QZXZQ.DAT. If there is a user-defined function in some programs (for example, a numerical integration program), the PIL program writes a FORTRAN subprogram and store it as QZXZQ.FOr.

(4) Using the PIL command EXECUTE or RUN, the PIL program automatically passes the control to the monitor to execute a designated program ENG:REALEQ.REL. If there is a QZXZQ.FOR prepared for a user-defined function, it is compiled and included. The PIL program function is now completed, and the execution file automatically takes over.

(5) The execution program ENG:REALEQ.REL reads the data and options stored in QZXZQ.DAT.

(6) Based on the input data and option selected, ENG:REALEQ.REL executes and outputs the results on the user's terminal.

(7) The Library program will generally allow the user to repeat the same problem but using a different method. If the user declines to repeat, the execution is concluded.

A memorandum by the author was distributed to the faculty and the students in 1969 concerning the Library. The memorandum has been revised and updated many times, most recently in 1980. The text of the memorandum of the most recent version is given in the following pages to complete the details of the Library.

University of Pittsburgh

SCHOOL OF ENGINEERING
Department of Electrical Engineering

MEMO TO: Faculty and Students, School of Engineering

FROM: T. W. Sze

DATE: June 1, 1969;
First Revision, November 1, 1970;
Second Revision, January 18, 1972;
Third Revision, July 1, 1974;
Fourth Revision, January 31, 1977;
Fifth Revision, September 1, 1980.

SUBJECT: Interactive Engineering Program Library on Device ENG:

INTRODUCTION

The use of computers in the Engineering curriculum is now a standard practice.

One serious problem, however, has always been the time-consuming work of preparing computer programs, debugging and executing them. Although libraries of subroutines of general interest are available, such as the IMSL package, the process of incorporating them in a course is a major task. It usually involves a search through a thick catalog for a program fitting the problem, learning the algorithm, finding out the particular input and output requirements and types. In some cases, special arrangements of large memory authorization, long execution time, or special periperal equipment are necessary in order to use these programs. Such laborious procedure has a discouraging effect to faculty and students using the computer effectively and extensively.

Thus, when the computer usage is included in a course, a very undesirable situation may sometimes emerge. Homework and projects can often be degenerated into long programming exercises that force the students (and instructors) to spend more time and efforts in getting their programs to run than to try to understand the course materials.

The Interactive Engineering Program Library was set up to overcome these limitations. In general, the Library will attempt to accomplish three objectives:

1. The programs are designed efficiently so that the core assignment and execution time stay within those authorized for the student users, even for relatively large size problems. Therefore, no special arrangement or authorization is necessary. With the easy use of the Library, the computer will indeed become an important day-to-day tool.

2. The user will not spend time in preparing programs. He will not be required to study lengthy program documentations; and thus there will be no distraction from the course materials.

3. When the time and effort on the "dog work" is drastically reduced (see the Appendix section of this memo), it will then be possible to upgrade the quality and the level of all engineering courses.

The Library is currently installed on-line in all three systems (A, B and C) and is given a device name of ENG: It contains a group of programs of general interest to engineering faculty and students. By man-machine interaction, a problem is shaped as the user specifies the data and the option. Suitable instructions and comments are printed out as prompting remarks along the way to guide the user. All programs, regardless of their programming languages, are written in "conversational mode" so that the user will be guided in how to use the program. Hence, it is not essential for a user of the Library to have any in-depth knowledge of the program, the language, or the algorithm once he learns the simple procedure of calling and executing the Library program.

Furthermore, these programs in the Library are mathematical-technique oriented rather than problem- and application-oriented. For example, a differential equation solution program can be used for circuit analysis, process control, vibration and stress analysis, material and energy balance equations for chemical dynamics, system stability studies, etc.

Although helpful, an in-depth knowledge of any programming or programming language is not a prerequisite to be a user of the Library.* A user will have an option to choose which algorithm and program for his problem. If he does not have any opinion, the program will pick up one that has proven general utility.

When a Library program begins its execution, the user's function will be to enter numerical data and to answer YES/NO to the computer's inquiries. When any input data are called for, prompting information of what sort of data and in what format will be printed out at the user's terminal to guide him. The user will then supply data as asked, or answer questions posed.

---

* As a matter of information, a Library program consists of an interactive problem-defining and input stage in PIL language, but will switch automatically to a REL or an EXE file for execution. The REL file or the EXE file is prepared from a FORTRAN program.

LIBRARY USAGE PROCEDURE

Currently, all Library programs are stored in the device "ENG:" in Systems A, B and C. Procedure for Library usage includes three simple steps:

Step 1: To get on the computer, either System A, B or C.

Step 2: To call and execute a chosen Library program.

Step 3: To get off the System.

If there is more than one problem or more than one run, Step 2 is repeated. thus, only the step-2 will be explained in some details.

To Call and Execute a Library Program

Suppose the name of the Library program chosen is WXYZ. Call and execute the program by entering a command*: (must be entered right after the prompting symbol).

```
.PIL ENG:WXYZ
```

The computer will respond with a printout of "READY:", and will then load the chosen program and start the execution. The memory requriement is self-adjusting (another built-in feature of the Library programs) and the user need not be concerned with it, unless he attempts to enter a problem too large for the program. Once the execution of a program begins, the user follows the printed instructions to enter numerical data and to answer YES-NO questions to complete the input phase, after which computer switches to a machine program execution to a completion.

---

* The old way of ".PIL WXYZ[33,33]" will still be valid for some period of time in order to allow orderly transition and revisions of departmental instructional materials.

A CONDENSED CATALOG OF THE LIBRARY

Basic mathematic techniques of the following areas are included in the Library:

Polynomial equation solution, real and complex roots.

Transcendental equation solution, real roots only.

Linear simultaneous equations, real or complex coefficients.

Basic real/complex matrix operations: +,-,* or inversion.

Generalized inverse of a matrix.

Other matrix operations, including determinants, Eigen values,state transition matrix, and characteristic equation.

Numerical integration, with user-specified accuracy.

Least square fits: linear, quadratic, cubic, and exponential fits.

Ordinary differential equations, first order and second order, linear or nonlinear.

Ordinary differential equations, nth order, max n=8.

Optimization of a nonlinear function, constrained or unconstrained

Linear programming

Fourier analysis

Fast Fourier Transform

Computer-aided logic design

Simulators and cross assemblers of microprocessors.

Graphic plots.

Course grade management (an information management system)

General utilities

In each of these, options of methods are available for the user's choice. At the conclusion of a solution, the programs are usually recycled so that the user may repeat the problem with a different method without repeating the input phase.

PROGRAM NAME	PROGRAM FUNCTIONS
BASMAT	Matrix operations to calculate any of the following: determinant, inverse, Eigen values, state transition matrix, characteristic equation, and state resolvent matrix.
BODE	To calculate and plot the frequency response (Bode diagram) from a transfer function given as a ratio of two polynomials. Output in tabular plus plot form.
CMIS	To prepare class roster, enter test grades, calculate test statistics, grade opti-scan exams, determine final term grades according to an instructor-specified formula.
COMEQ	Solution for linear simultaneous equations with complex coefficients, 10 complex unknowns maximum, double-precision computation. Crout's elimination method.
COGGIN	Coggin's method for maximum/minimum search of a single variable function, unconstrained.
CSMP	To prepare a control file and submit automatically as a batch job.
DF1	Solution of first order ordinary differential equation. Algorithm options: (1) Modified Euler's method (2) Runge-Kutta method, 4th order (3) Milne's method (4) Adam-Moulton method (5) Hamming's method Output print options: (1) Output in tabular form only (2) Output in tabular form and plots
DF2	Solution of a second order ordinary differential equation. Same algorithm and output options as the program DF1.
DFN	Solution of an nth order (max n=8) ordinary diff equation. Algorithm options: (1) Modified Euler's method (2) Runge-Kutta method, 4th order (3) Adam-Moulton method Output print options: (1) Output in tabular form only (2) Output in tabular form and plots
DIRECT	To type out the most recent directory of the Library.
EZLP	To solve student-oriented linear programming problems. Help file available as ENG:EZLP.HLP

FFT	Fast Fourier Transform for a set of samples, using the Cooley-Tukey algorithm.
FIBON	Minimization of a single variable, nonlinear function by Fibonacci search algorithm.
FIT	Least square fit for n data points. Options available are: (1) Linear fit: y = a*x + b (2) Quadratic fit: y = a*x**2 + b*x +c (3) Cubic fit: y = a*x**3 + b*x**2 + c*x +d (4) Exponential fit: y = a + b*exp(c*x)
FOUR	Fourier analysis on a periodic waveform.
HELP	To print out a copy of the file ENG:ENG.HLP
HOOKE	Hooke-Jeeves method of pattern search optimization
IMAGE	Utility package of image processing to a standard 128x128x8 image file. Options include image print, image pixel value listing, transpose, linear combination, noise mixing. More options are in preparation.
M8080	A simulator for a multi-processor system employing INTEL 8080's. Help file available as ENG:M8080.HLP
MATOP	Basic operations for real or complex matrices: addition, subtraction, multiplication and inversion.
MINILP	Linear programming, interactive input phase.
MUX	A computer-aided logic design program of using multiplexer IC chip in a combinational circuit design.
NI	Numerical integration, Simpson's Rule, with user specified and controlled accuracy.
PLOT	To plot a curve on rectangular coordinates. Input options for y=f(x): (1) f(x) to be specified by the user as a FORTRAN expression. (2) f(x) data points already stored as a disk file. (3) f(x) data points to be entered via the terminal. Output options: (1) Output on the terminal or the line printer. (2) Output from the Calcomp plotter.
POLY	Real and complex roots of a polynomial equation. Algorithm options (1) Mairstow's method (2) Modified Newton-Raphson's method (3) Lin's method Normally, option 1 is recommended.

QUINE	Quine-McCluskey's method of Boolean function minimization, 12 variables maximum. Output options of either summation-of-products, or product-of-sums, or both.
REALEQ	Linear simultaneous equations with real coefficients, 30 variables maximum. Double-precision calculations in all cases. Algorithm options: (1) Gauss elimination (2) Gauss-Seidel iteration (3) Matrix inversion (4) Crout's elimination (5) Cramer's rule Input data may be entered via a disk file or the terminal.
RECO	Generalized inverse of a matrix with real elements. Maximum number of rows is 9.
S8080	A single INTEL 8080 simulator program, including a built-in editor. Help file available as ENG:S8080.HLP.
SCAMP	A simulator and cross assembler for the National Semiconductor SC/MP microprocessor trainers. Help files available as ENG:SCAMP1.HLP and ENG:SCAMP2.HLP.
SEARCH	An optimization package of search methods to find the optimal values of a constrained or unconstrained, single or multiple-variable, nonlinear function.
STATUS	Same as the program DIRECT
TRANEQ	Newton-Raphson method of transcendental equation solution, real root only. Use Stirling formula for numerical differentiation.
TRUTH	To generate a truth table, or a Karnaugh map, from a given Boolean function in the form of summation-of-products or product-of-sums.
CALPLT	To plot a curve on the CalComp plotter.

FORTRAN-CALLABLE SUBROUTINE LIBRARY

A group of FORTRAN-callable subroutine packages are also available in the device ENG: Their names and their current status are outliend below:

EE45 Subroutines from the EE45 text: COMPUTER METHODS FOR MATHEMATICAL COMPUTATIONS, by G. E. Forsythe, M. A. Malcolm, and C. B. Moler, Prentice-Hall Inc., 1977.

IMPROC An image processing package, including image filtering, transformation, transpose, etc. Applied to 128x128x8 image file size. Developed by T. W. Sze. Help file available as ENG:IMPROC.HLP.

SUBSET A utility subroutine set developed by Ronal K. Nicholas, included in the Engineering Library by permission. For reference, see: SUBSET MANUAL, by Ronal K. Nicholas, University of Pittsburgh, 1977 (available at the Book Center).

SIPROC A group of signal processing subroutines. For reference, see: PROGRAMS FOR DIGITAL SIGNAL PROCESSING, Edited by the Digital Signal Processing Committee, IEEE Acoustic, Speech, and Signal Processing Society. Published by IEEE Press, Institute of Electrical and Electronics Engineers, New York, 1979.

GRAPH A group of data plotting and tabulation subroutines. For reference, see ENG:GRAPH.HLP.

ACKNOWLEDGEMENTS

The Engineering Program Library project was initiated in 1969 when I developed and taught the course ES2 (Engineering Analysis II). Later, the project was maintained and expanded with the help of the Engineering staff of the Benedum RJE site. I wish to ackowledge the assistance rendered me by Mr. Wayne Baughman, and many graduate assistants involved in this project, particularly Drs. Richard Hsia, M.S. Nataraja, A.R. Modarressi, K.D. Oka, and Messrs. T. Goss and H.R. Anada. Credits are also due to Mr. Frank Heyn, who adapted the IEEE tape and modified it into the SIPROC package of the Library. Last but not the least, assistance is acknowledged to Mr. Michael A. Matzek of the Pitt Computer Center to shift the old call-PPN of [33,33] into the device of "ENG:"

The full content of this memo, minus the Appendix, is stored as a disk file, and may be reproduced on a line printer by a command:

```
.Q ENG:MEMO.DOC
```

APPENDIX ILLUSTRATIVE EXAMPLES

Several examples are included here to illustrate the procedure of using the LIBRARY programs. On the reproduction of the printouts, those text that were typed by the user were underlined, and those typed by the terminal were not. In addition, comments are added as brief explanations.

Example 1 Given the following polynomial equation, find all roots, real and complex:

$$x^7 - 80x^6 + 20x^5 - 2200x^4 + 1350x^3 + 1350x^2 - 1200x + 800 = 0$$

The program chosen was POLY, and all three methods were run. The modified Newton's method turned out to be divergent, but the other two methods worked okay for this problem. It took 3 minutes on the terminal for this problem.

Example 2 Solve for the solution of a system of simultaneous equations with complex number coefficients. The equations in matrix form are:

$$\begin{bmatrix} 20+j30 & -5-j3 & -15+j16 & 0+j0 & 0-j43 \\ -5-j3 & 11-j3 & 0+j0 & -6+j8 & 0+j0 \\ -15+j16 & 0+j0 & 42-j32 & 8+j16 & -15+j10 \\ 0+j0 & -6+j8 & -12+j16 & 23-j42 & -5+j18 \\ 0-j43 & 0+j0 & -15+j10 & -5+j18 & 20+j15 \end{bmatrix} \cdot \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \end{bmatrix} = \begin{bmatrix} 100+j0 \\ 200+j100 \\ 0+j0 \\ 0+j0 \\ 0+j0 \end{bmatrix}$$

Observe particularly the error-recovery procedure built in the program COMEQ. When certain input data were incorrectly entered, the program allows the user to check the data and make corrections. This avoids repeating the input of large amount of data, which would have to be done if the problem must be aborted.

Example 3 Plot a curve represented by

$$y = .359\, e^{-0.1x} \sin(0.1x) \sin(0.5x)$$

for the range from x=0 to x=30 with an increment of unity. The program PLOT was used for this example, and the plot was reproduced on the terminal. Total time consumed on the terminal for this problem was 6 minutes. The same problem, using an increment of 0.1, was also plotted on the Calcomp plotter, and the result is also shown.

Example 1

```
LOGIN 115103/132311                                   Step 1: To get on the system
JOB 31 PITT DEC-1077/B 61B,72A TTY63
Password:
xxxx units remaining   Last login: 1112 4-Jan-77
1120    7-Jan-77          Fri

.PIL ENG:POLY                                         Step 2: To call and execute an
Proc01:                                                       IPL program.

POLYNOMIAL EQUATION, INTERACTIVE PROGRAM
LOADING THE PROGRAM...

ENTER THE ORDER OF POLYNOMIAL EQUATION (N):
N =>2

ENTER IN THE ORDER OF DESCENDING POWERS, COEFFICIENTS OF EACH TERM
ENTER 0 FOR MISSING TERM.   USE FREE FORMAT
=>1  80  20  -2200  1350  1350  -1200  800

ENTER ALL COEFFICIENTS:
A( 1) =      1.0000
A( 2) =     80.0000
A( 3) =     20.0000
A( 4) =  -2200.0000
A( 5) =   1350.0000
A( 6) =   1350.0000
A( 7) =  -1200.0000
A( 8) =    800.0000

ARE ALL COEFFICIENTS OK?
YES OR NO?  =>YES

THREE METHODS ARE AVAILABLE:
     OPTION 1 = BAIRSTOW'S METHOD
     OPTION 2 = MODIFIED NEWTON'S METHOD
     OPTION 3 = LIN'S METHOD
UNDER NORMAL CIRCUMSTANCES, OPTION 1 IS RECOMMENDED.
OPTION = >1
LOADING
PGM 3K CORE
EXECUTION
```

To enter data and answer questions.

Underlined texts are data or answer typed by the user.

Switching automatically to machine language program execution

SOLUTION BY BAIRSTOW'S METHOD:

ROOT NO.	REAL PART	IMAG PART
1	-0.9664E+00	
2	0.1011E+01	
3	0.2817E+00	0.5313E+00
4	0.2817E+00	-0.5313E+00
5	-0.3493E+00	0.5306E+01
6	-0.3493E+00	-0.5306E+01
7	0.8009E+02	

Results

```
WANT TO TRY ANOTHER METHOD FOR THE SAME PROBLEM?
IF YES, TYPE NEW OPTION NUMBER.   IF NO, TYPE 0:
>2

NEWTON'S METHOD IS DIVERGENT.  TRY ANOTHER METHOD.

WANT TO TRY ANOTHER METHOD FOR THE SAME PROBLEM?
IF YES, TYPE NEW OPTION NUMBER.   IF NO, TYPE 0:
>3

SOLUTION BY LIN'S METHOD:
```

Option to repeat the problem

ROOT NO.	REAL PART	IMAG PART
1	0.2817E+00	0.5313E+00
2	0.2817E+00	-0.5313E+00
3	-0.9664E+00	
4	0.1011E+01	
5	-0.3492E+00	0.5306E+01
6	-0.3492E+00	-0.5306E+01
7	0.8009E+02	

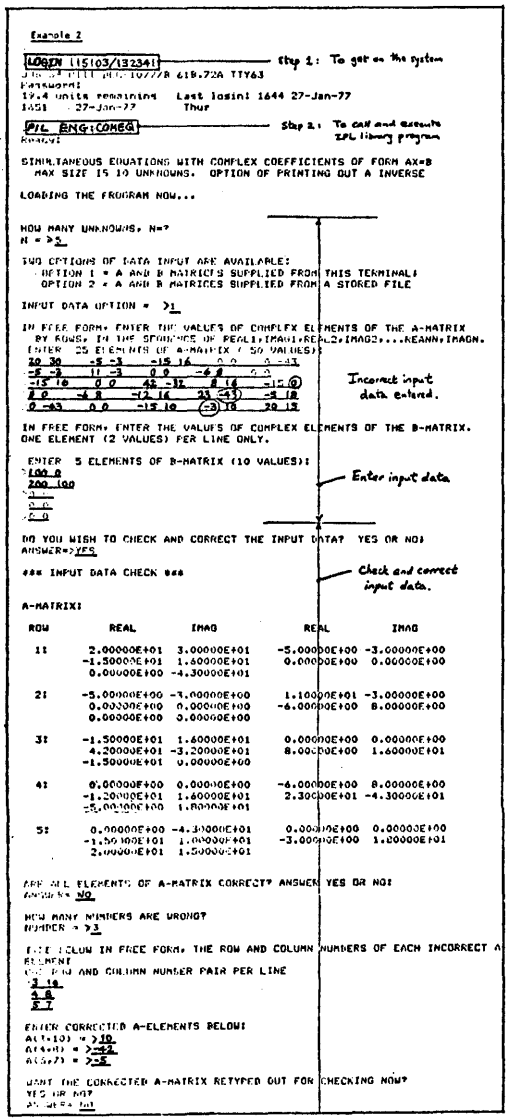
```
WANT TO TRY ANOTHER METHOD FOR THE SAME PROBLEM?
IF YES, TYPE NEW OPTION NUMBER.   IF NO, TYPE 0:
>0
STOP

END OF EXECUTION
CPU TIME: 1.08  ELAPSED TIME: 43.23
EXIT
```

```
.BYE                                                  Step 3: To get off the system
Job 31 [115103,132341] off TTY63  at 1122 7-Jan-77  Connect=3 Min
Disk R+W=210+33  Tape IO=0  Saved all files (12 blocks)
CPU 0:04  Core HWM=10P  Units=0.0130 ($0.98)
```

Three methods were tried for a degree-7 polynomial equation.

Total time at the terminal = 3 minutes

Computer time used = 4 seconds



```
Example 2

LOGIN [115103/132341]                    Step 1: To get on the system
Password:
19.4 units remaining   Last login: 1644 27-Jan-77
1451   27-Jan-77          Thur
PIL ENG:COMEG                            Step 2: To call and execute
Ready:                                   IPL library program

SIMULTANEOUS EQUATIONS WITH COMPLEX COEFFICIENTS OF FORM AX=B
  MAX SIZE IS 10 UNKNOWNS.  OPTION OF PRINTING OUT A INVERSE

LOADING THE PROGRAM NOW...

HOW MANY UNKNOWNS, N=?
N = >5

TWO OPTIONS OF DATA INPUT ARE AVAILABLE:
   OPTION 1 = A AND B MATRICES SUPPLIED FROM THIS TERMINAL
   OPTION 2 = A AND B MATRICES SUPPLIED FROM A STORED FILE

INPUT DATA OPTION =  >1

IN FREE FORM, ENTER THE VALUES OF COMPLEX ELEMENTS OF THE A-MATRIX
  BY ROWS, IN THE SEQUENCE OF REAL1,IMAG1,REAL2,IMAG2,...,REALN,IMAGN.
ENTER  25 ELEMENTS OF A-MATRIX ( 50 VALUES):
20 30    -5 -3    -15 16    0 0    0 -33
-5 -3    11 -3    0 0    -6 8    0 0
-15 16   0 0    42 -32    8 16    -15 0
8 0    -6 8    -12 16    25 -43    -9 18
0 -33    0 0    -15 10    -9 18    20 15

IN FREE FORM, ENTER THE VALUES OF COMPLEX ELEMENTS OF THE B-MATRIX.
ONE ELEMENT (2 VALUES) PER LINE ONLY.

 ENTER  5 ELEMENTS OF B-MATRIX (10 VALUES):
>100 0
200 100
>1
>0
>0

DO YOU WISH TO CHECK AND CORRECT THE INPUT DATA?  YES OR NO:
ANSWER=>YES

### INPUT DATA CHECK ###

A-MATRIX:

 ROW        REAL          IMAG          REAL          IMAG
  1:   2.00000E+01   3.00000E+01   -5.00000E+00  -3.00000E+00
      -1.50000E+01   1.60000E+01    0.00000E+00   0.00000E+00
       0.00000E+00  -4.30000E+01

  2:  -5.00000E+00  -3.00000E+00    1.10000E+01  -3.00000E+00
       0.00000E+00   0.00000E+00   -6.00000E+00   8.00000E+00
       0.00000E+00   0.00000E+00

  3:  -1.50000E+01   1.60000E+01    0.00000E+00   0.00000E+00
       4.20000E+01  -3.20000E+01    8.00000E+00   1.60000E+01
      -1.50000E+01   0.00000E+00

  4:   0.00000E+00   0.00000E+00   -6.00000E+00   8.00000E+00
      -1.20000E+01   1.60000E+01    2.30000E+01  -4.30000E+01
      -5.00000E+00   1.80000E+01

  5:   0.00000E+00  -4.30000E+01    0.00000E+00   0.00000E+00
      -1.50000E+01   1.00000E+01   -3.00000E+00   1.80000E+01
       2.00000E+01   1.50000E+01

ARE ALL ELEMENTS OF A-MATRIX CORRECT?  ANSWER YES OR NO:
ANSWER= NO

HOW MANY NUMBERS ARE WRONG?
NUMBER = >3

ENTER (ROW,COLUMN) IN FREE FORM, THE ROW AND COLUMN NUMBERS OF EACH INCORRECT A
ELEMENT
ONE ROW AND COLUMN NUMBER PAIR PER LINE
>3 10
>4 8
>5 7

ENTER CORRECTED A-ELEMENTS BELOW:
A(I,J) = >10
A(J,I) = >-32
A(J,J) = >-5

WANT THE CORRECTED A-MATRIX RETYPED OUT FOR CHECKING NOW?
YES OR NO?
ANSWER= NO
```

Example 2

Step 1: To get on the system

Step 2: To call and execute IPL library program

Incorrect input data entered.

Enter input data

Check and correct input data.

```
B-MATRIX CHECK:

 ROW        REAL          IMAG
  1:   1.00000E+02   0.00000E+00
  2:   2.00000E+02   1.00000E+02
  3:   0.00000E+00   0.00000E+00
  4:   0.00000E+00   0.00000E+00
  5:   0.00000E+00   0.00000E+00

ARE ALL B-ELEMENTS CORRECT?  ANSWER YES OR NO:
ANSWER=>YES

DO YOU WANT THE INVERSE OF A-MATRIX PRINTED OUT?  YES OR NO?
ANSWER=>NO

NOW SWITCHING TO FORTRAN EXECUTION...

LOADING

COMEG 3K CORE
EXECUTION

A-MATRIX:

 ROW  1:    0.2000E+02  0.3000E+02    -0.5000E+01 -0.3000E+01
           -0.1500E+02  0.1600E+02     0.0000E+00  0.0000E+00
            0.0000E+00 -0.4300E+02

 ROW  2:   -0.5000E+01 -0.3000E+01     0.1100E+02 -0.3000E+01
            0.0000E+00  0.0000E+00    -0.6000E+01  0.8000E+01
            0.0000E+00  0.0000E+00

 ROW  3:   -0.1500E+02  0.1600E+02     0.0000E+00  0.0000E+00
            0.4200E+02 -0.3200E+02     0.8000E+01  0.1600E+02
           -0.1500E+02  0.1000E+02

 ROW  4:    0.0000E+00  0.0000E+00    -0.6000E+01  0.8000E+01
           -0.1200E+02  0.1600E+02     0.2300E+02 -0.4300E+02
           -0.5000E+01  0.1800E+02

 ROW  5:    0.0000E+00 -0.4300E+02     0.0000E+00  0.0000E+00
           -0.1500E+02  0.1000E+02    -0.5000E+01  0.1800E+02
            0.2000E+02  0.1500E+02

B-MATRIX:

 ROW  1:    0.1000E+03  0.0000E+00

 ROW  2:    0.2000E+03  0.1000E+03

 ROW  3:    0.0000E+00  0.0000E+00

 ROW  4:    0.0000E+00  0.0000E+00

 ROW  5:    0.0000E+00  0.0000E+00

SOLUTION OF THE COMPLEX SIMULTANEOUS EQUATIONS:

                 REAL PART    IMAGINARY PART   MAGNITUDE   PHASE ANGLE(DEG)
 X( 1) =      11.579755      -3.28194        12.03586      -15.82
 X( 2) =      27.542859       8.38943        28.79221       16.94
 X( 3) =       7.869136      -7.04257        10.56035      -41.83
 X( 4) =      12.317659      -1.68885        12.43290       -7.81
 X( 5) =       9.040064      -2.60817         9.40879      -16.09

STOP

END OF EXECUTION
CPU TIME: 0:00  ELAPSED TIME: 55.85
EXIT
```

Automatic linkage from PIL to FOR execution

Printout of problem conditions

Results

Step 3: To get off the system.

```
/off
JOB 36 [115101,132141] off TTY63  at 1656 27-Jan-77  Connect=6 Min
Disk R+W=14951  Tape IO=0  Saved all files (10 blocks)
CPU 0:07  Core HWM=12P  Units=0.0170 ($1.29)
```

Total time at the terminal = 6 minutes

Computer time used = 7 seconds

Example 3

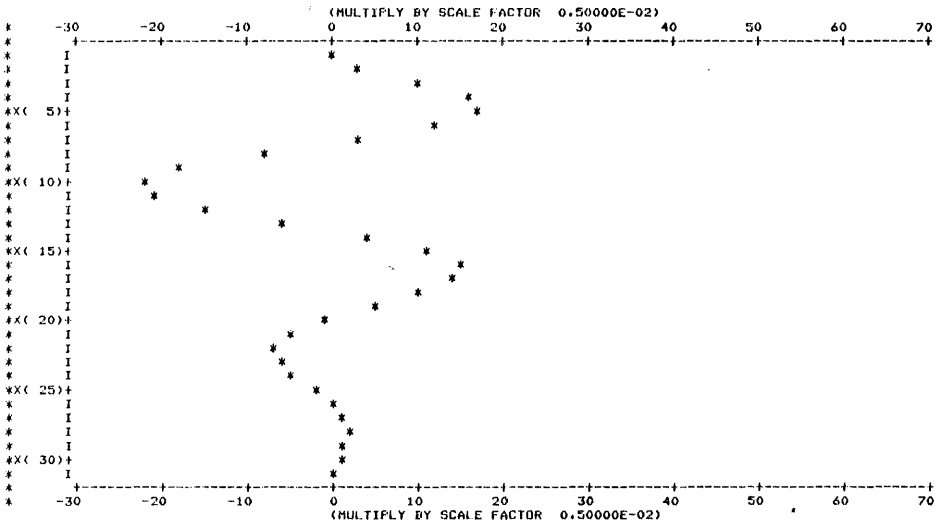
```
LOGIN 115103/13234T                          Step 1: To get on the system
JOB 28 PITT DEC-1077/B 61B.72A TTY63
Password:
1?.4 units remaining   Last login: 1451 27-Jan-77
153A    27-Jan-77       Thur

PIL ENG:PLOT                                 Step 2: To call and execute
Ready:                                                the IPL program

INTERACTIVE PLOTTING PROGRAM .
MAXIMUM CAPACITY: PLOTTING A CURVE OF 151 DATA POINTS.
LOADING THE PROGRAM NOW,...


THERE OPTIONS ARE AVAILABLE:
   OPTION 1 = TO PLOT A CURVE FOR Y=F(X) WHICH WILL BE SUPPLIED BY YOU,
   OPTION 2 = TO PLOT A CURVE WITH COORDINATES ALREADY STORED ON FILE,
   OPTION 3 = TO PLOT A CURVE WITH DATA TO BE ENTERED AT THIS TERMINAL.

OPTION = >1

TWO TYPES OF PLOTS ARE AVAILABLE:
   TYPE 1 = PLOT PRODUCED ON TERMINAL OR LINE PRINTER
   TYPE 2 = PLOT PRODUCED ON CALCOMP PLOTTER

YOU CAN OBTAIN THE PRINTER OR TTY PLOT RIGHT AFTER EACH RUN, BUT YOU
MUST WAIT UNTIL THE NEXT DAY TO GET THE CALCOMP PLOT.   ON THE OTHER
HAND, CALCOMP PLOT IS OF MUCH SUPERIOR QUALITY.

CHOOSE THE TYPE OF PLOT YOU WANT:
TYPE = >1

HOW MANY POINTS TO BE PLOTTED (NPT=?)?
DO NOT CONFUSE NUMBER OF INCREMENTS WITH NUMBER OF POINTS.   IF YOU
HAVE 100 INCREMENTS, YOU SHOULD SPECIFY 101 POINTS, AS AN EXAMPLE.

NPT = >31

WHAT IS THE VALUE OF THE FIRST X (X1=?) ?
X1 = >0

WHAT IS THE VALUE OF THE LAST X (X2=?) ?
X2 = >10

TYPE THE FORTRAN EXPRESSION OF F(X), SUCH AS:
      3.12*EXP(-3.5*X)*SIN(377.*X)

REMEMBER: FORTRAN EXPRESSION, NOT PIL EXPRESSION.

ENTER F(X)
F(X)=  >0.359*EXP(-.1*X)*SIN(0.1*X)*SIN(0.5*X)
```

```
NOW READY TO SWITCH TO FORTRAN EXECUTION...

FORTRAN: OZXZQ
FCN
LOADING

PLOT 6K CORE
EXECUTION


TYPE THIS COMMAND TO RECEIVE YOUR OUTPUT ON LINE PRINTER:-

    Q PLOT.LPT/FILE:FORTRAN

TYPE THIS THIS COMMAND TO RECEIVE OUTPUT ON YOUR TERMINAL:-
    TTY WIDTH 132
    TYPE PLOT.LPT

YOU MUST GET YOUR OUTPUT BEFORE NEXT PLOT RUN.
OTHERWISE, THIS OUTPUT WILL BE ERASED AND REPLACED BY NEW PLOT OUTPUTS.


STOP

END OF EXECUTION
CPU TIME: 0.73  ELAPSED TIME: 4.53
EXIT
```

```
.TTY WIDTH 132
.TYPE PLOT.LPT
```

} Commands to reproduce the plot on the terminal

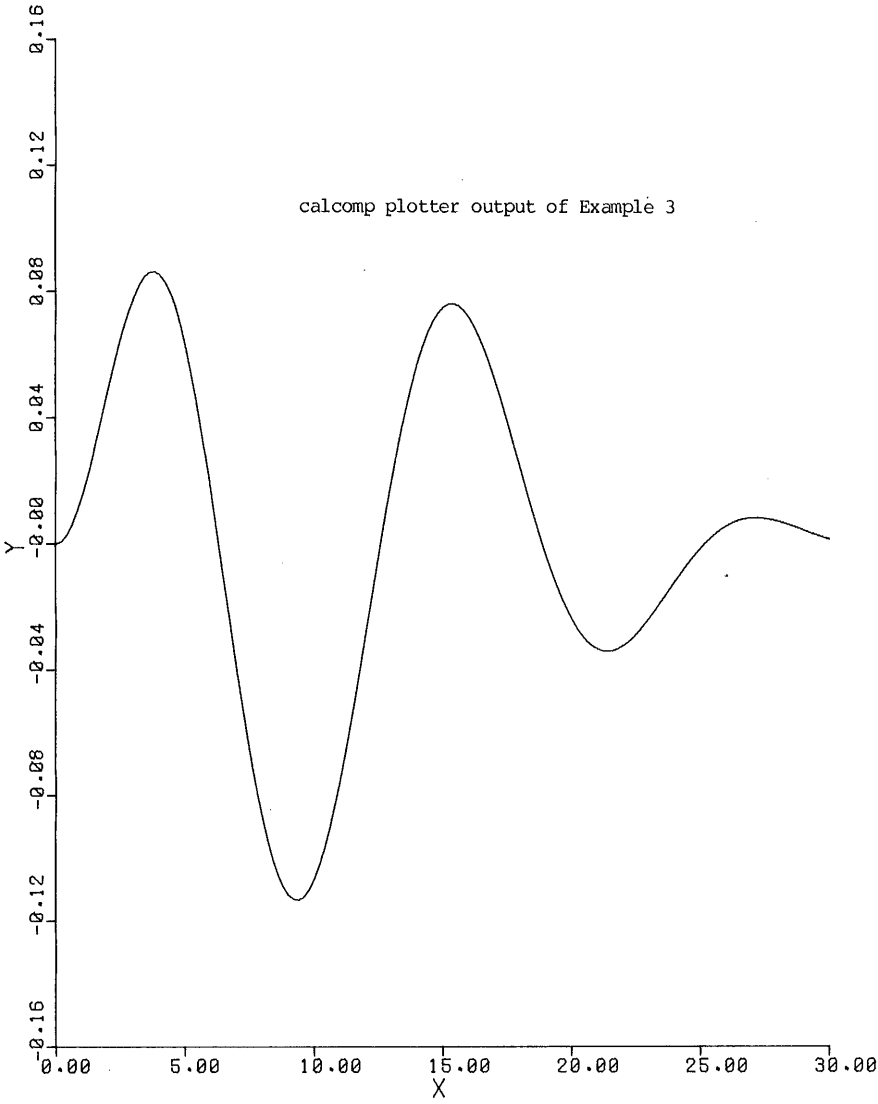
```
THE SCALE FACTOR OF ORDINATE: 1 DIVISION= 0.50000E-02

THE SCALE FACTOR OF ABSCISSA: 1 DIVISION= 0.10000E+01

        FIRST ABSCISSA VALUE        X(1)= 0.00000E+00

  NOTE: IN INTERPRETING THE PLOT, X-AXIS STARTS WITH X(1) VALUE.
SUBSEQUENT X CAN BE COMPUTED FROM X(1) AND ABSCISSA SCALE FACTOR.
```




```
.
.BYE
Job 28 [115103,132341] off TTY63  at 1541 27-Jan-77  Connect=6 Min
Disk R+W=234+57  Tape IO=0  Saved all files (18 blocks)
CPU 0:08  Core HWM=16P  Units=0.0195 ($1.46)
```

Step 3 : To get off the system.

Total Terminal time = 6 minutes
Total computer time = 8 seconds


calcomp plotter output of Example 3
0.16
0.12
0.08
0.04
-0.00
-0.04
-0.08
-0.12
-0.16
Y
0.00
5.00
10.00
15.00
20.00
25.00
30.00
X

EXAMPLE 3

A

B

C

D

E

F

G

H

I

K

L

M

O

P

Q

R

S

T

U

V

W

$

INDEX B QUICK REFERENCE OF COMMANDS AND PROGRAMS

This INDEX includes a list of commands or subprograms for rapid reference. The legend of entries is:

Command or Subprogram Name (Processor Name), Page number

For example, the entry "SCALE (CALCOMP), 252" means a subprogram named SCALE for the CalComp Plotter processor, and its description may be found on page 252. The following processors are included in the Quick Reference:

BATCH	Commands in the Batch processor
CALCOMP	Subprograms in the CalComp Plotter Subprogram Package PRG:PLTLIB.REL
Engineering Library	Library programs in the device ENG:
FORTRAN	Subroutines in the System FORTRAN Library
MONITOR	Commands in the System Monitor
PIL	Commands in the PIL Processor
PLOT10	Subroutines in the Tektronix Graphic Package PLOT10
SUBSET	Subprograms of the SUBSET package, developed by Ronal K. Nicholas.
UPDATE	Commands in the UPDATE text editor
USL	User Program Library

A

B

C

D

E

F

G

H

I

J

K

L

M

N

O

P

Q

R

S

T

U

V

W

X

Z