PDP-10 REFERENCE HANDBOOK

Prepared by
The Software Writing Group
Programming Department
Digital Equipment Corporation

Additional copies of this handbook may be ordered from the Program Library, DEC, Maynard, Mass. 01754. Order code: AIW. $5.00 each. Discounts available on five or more copies.

PDP-10 HANDBOOK SERIES

The material in this handbook, including but not limited to instruction times and operating speeds, is for information purposes and is subject to change without notice.

FOREWORD

One of the significant measures of the quality of a computing system like the PDP-10 is the utility and availability of systems documentation. Good manuals are the vital communications link between DEC and the people who use our systems.

This collection has been organized for the convenience of PDP-10 programmers, systems analysts, engineers and others who work at the machine language level.

I'm pleased to acknowledge here the work of the many DEC system designers, engineers, and system programmers who continue to advance the state of the time-sharing art in both hardware and software. Also, to our PDP-10 users, who during the past two years, have helped immeasurably to improve and refine the system, and to the DEC software writers and technical artists who prepared this volume, our special thanks.

Kenneth H. Olsen

President, Digital Equipment Corporation

PREFACE

This volume is a comprehensive library of information for experienced programmers, systems analysts, and engineers who are interested in writing and operating assembly-language programs in the PDP-10 time-sharing environment.

The first three chapters deal with program preparation. Chapters 4 and 5 are about loading, editing, testing, debugging, and running source language programs. Chapter 6 contains a miscellaneous collection of utility programs that have proven most useful to system designers and experienced programmers.

As we expect to improve this book in future revisions, all readers are earnestly requested to send corrections and comments to:

Manager, Software Writing Group
Programming Department
Digital Equipment Corporation
Maynard, Mass. 01754

A companion volume for beginning programmers and others who prefer to write programs in one of the popular compiler or conversational languages is scheduled for publication in 1970.

CONTENTS

Foreword III

Preface IV

System Overview VI

Introduction XI

Book 1 Programming with the PDP-10 Instruction Set 1

Description of the Central Processor Structure, General Word Format, Memory Characteristics, and Assembler Source-Programming Conventions, Followed by a Presentation of the Specific Instruction Format, Mnemonic and Octal Op Codes, Functions and Timing Formulas.

Book 2 Assembling the Source Program 187

Reference Information for the PDP-10 Assembly System, containing Explanation of Format of Statements, Use of Pseudo-Operations and the Coding of Macro Instructions.

Book 3 Communicating with the Monitor 283

Complete Guide to the Use of the Time-Sharing Monitors: Monitor Commands, Allocation of Facilities, Relocation and Protection of User Programs, and Description of the Reentrant Capability.

Book 4 Editing the Source Program 491

Procedures for Creating, Modifying and Displaying Source Files Recorded in ASCII Characters.

Book 5 Executing the Program On-Line 525

Description of Loading and Linking of Relocatable Binary Programs Generated by MACRO-10 or FORTRAN IV and Exposition of Commands and Techniques for On-Line Checkout and Testing of MACRO-10 and FORTRAN IV Programs.

Book 6 Utility Programs 583

Transferring Data Files Between Standard Devices, Updating Files Containing Relocatable Binary Programs, Manipulating Programs within Program Files, Cross-Referencing User Defined Operators, Op Codes, Pseudo-Op Codes and Global Symbols, Comparing Source Files and Comparing Binary Files, Serving and Restoring Core Images on DECtape.

Appendices 629

Master Index/Glossary 637

.SYSTEM OVERVIEW

The PDP-10 is the successful culmination of many years of computer design research — a process which has enabled Digital Equipment Corporation to provide better computers at the lowest possible prices.

Starting with the PDP-1 in 1959, DIGITAL has pioneered the development of real-time systems for science and industry. Since then, each new system has increased in versatility, yet has consistently decreased the cost of computation. The PDP-1 was the first powerful real-time computer for under $150,000. The PDP-8 showed that an effective computer could sell for less than $20,000, and newer models in the PDP-8 family have lowered the cost to less than $10,000.

In developing its time-sharing capability, DIGITAL has built a history of success very similar to the company's record in real-time applications. DIGITAL's customers have been building time-sharing systems around PDP computers since 1960. And, in 1963, DIGITAL developed its own time-sharing computer, PDP-6 — the first to be delivered with manufacturer-supplied hardware and software.

The PDP-10 reflects DIGITAL experience in both real-time and time-sharing. The system performs time-sharing and real-time operations equally well and simultaneously and provides concurrent batch processing.

In conversational time-sharing, up to 63 users at local and remote locations can simultaneously develop programs on remote consoles and receive answers to mathematical or engineering problems in seconds. PDP-10 time-sharing monitors provide instantaneous response for the users so that they can perform on-line composition, editing, and debugging of programs in FORTRAN IV, MACRO-10, COBOL, BASIC, and AID, with the use of EDITOR, TECO and DDT. The monitors can handle any mixture of these languages and programs concurrently. And most of the software is re-entrant so that multiple users can share the same compiler or utility program for increased efficiency.

For programs that don't require immediate processing, users may initiate batch processing — a task which proceeds concurrently with time-sharing and real-time tasks. In batch processing, the PDP-10 can handle any stream of programs, such as a mixture of FORTRAN, MACRO-10 and COBOL. Normally, batch processing operates without operator attention. However, the PDP-10 allows the operator to stop or start the batch system, rearrange the queue or call for a print-out to analyze program errors. The operator can also select and assign the desired input, output, and temporary storage devices to be used in batch processing.

When real-time operations such as data acquisition and control are the primary purpose of the PDP-10, system software provides response in microseconds, processing information at speeds that meet the most demanding requirements. High priority real-time tasks are interfaced directly to the priority interrupt system and control their own input/output operations for unlimited flexibility. Less critical real-time jobs are monitor controlled with a real-time clock assuring that each task does not exceed its allotted time and destroy the response of other programs. For even greater efficiency, time not used by the real-time programs can be used for conversational time-sharing and/or batch processing.

Structure of the PDP-10

Every PDP-10 uses one of three levels of monitors to allocate resources and perform input/output functions. The single-user monitor is used for dedicated systems which operate one program at a time. The multi-programming monitor controls the execution of multiple programs in core, switching between them at microsecond speeds. The swapping monitor effectively increases the available core by swapping programs between high speed disk or drum storage and core memory. Thus more users can be served by a given amount of core. All language processors (FORTRAN, MACRO-10, COBOL, BASIC, and AID) operate identically under the multi-programming and swapping monitors.

To make efficient use of memory, language processors and important utility programs are re-entrant, that is, the pure code for each program can be shared by multiple users. Re-entrancy is possible since any program may be separated into a pure segment that never requires modification, and a segment which contains code or data which is relevant only to a particular user. For example, a re-entrant system can service three FORTRAN users with one 8 K compiler (pure code) and three 2 K user areas, a total of 14 K, whereas a non-re-entrant system would require 30 K for the same programs. Since more users can occupy a given amount of core space, system response improves and swapping time is reduced. Dual protection and relocation registers protect the active user and allow the program segments to reside in two non-contiguous sections of core memory.

The PDP-10 has a 36-bit word length allowing it to store 25 to 30 percent more information than a 32-bit system. The system stores five 7-bit USACII characters whereas the smaller word length computer stores only four characters. It also provides more accuracy in single precision floating arithmetic than computers with 32-bit word size.

The PDP-10 has a large instruction repertoire to simplify assembly coding and reduce the size of higher level programs. Its 366 instructions divide logically into families and are easily learned. The list also includes an extensive set of floating point and byte manipulation instructions. Due to instruction set efficiency, fewer instructions are required to perform a given function. Assembly language programs are therefore shorter than with other computers and the instruction set simplifies monitor systems, language processors, and utility programs. For example, compiled programs are 30 to 50 per cent shorter, require less memory, and execute faster than those of comparable computers.

Sixteen high speed integrated circuit registers also help improve program execution. Depending on program requirements, these registers can serve as accumulators, normal memory location, and/or index registers. Intermediate results of computations are stored in the registers rather than in core memory; thus, no instructions are needed to store and retrieve the data and data is available in nanoseconds. Fifteen of the registers can be used as fast memory locations so that program segments with sixteen or fewer instructions can be executed repetitively at very high rates.

The PDP-10 memory bus structure gives the central processor and high speed data channels simultaneous access to separate memory modules. Only when the processor and a data channel access the same module does the processor lose a memory cycle. Modules contain up to four ports, allowing access to a total of four processors and/or channels. Such parallel operation improves processor utilization, yielding manyfold improvements over systems which provide only a single path to memory. The bus system allows each data channel to transmit full 36-bit words at speeds of up to one million words (5 million 7-bit characters) per second.

Memory can be modularly expanded to 262,144 words of core, all of which (including the 16 accumulators and 15 index registers) can be directly addressed. Total memory capacity can be comprised of combinations of modules in 8,192-, 16,384-, 32,768-, 65,356-, and 131,072-word blocks. Memory banks are asynchronous allowing interleaving and making it possible to intermix memories of different cycle times.

To provide immediate service to real-time requests, the PDP-10 has a multi-level priority interrupt system. The system is a hardware feature, but is programmable for increased flexibility. Devices may be assigned to any level under program control and the entire interrupt system or any level may be selectively turned on or off.

PDP-10 Configurations

The modularity of PDP-10 hardware and software makes it economical to configure a wide variety of systems and easy to expand the systems in the field. (See PDP-10 hardware list in Appendix A.) An individual can buy a single user system and, at some later date, expand to a small time-sharing system. Or the small time-sharing user can expand his system to serve 63 users simultaneously. Within any basic configuration, the user has a wide choice of memory sizes and speeds, input/output equipment, and storage facilities. For example, the input/output system alone can accommodate up to 128 discrete devices and device controllers, permitting almost limitless expansion of on-line storage and other input/output equipment.

The single-user system in Figure 1 can be simple or as elaborate as the user requires. As shown, it consists of an arithmetic processor, one or more core memory modules, a DECtape control and DECtape units[1], a console teletype, and a paper tape reader and punch. By adding more core memory and data line scanner, the system can easily be converted to multi-programming.

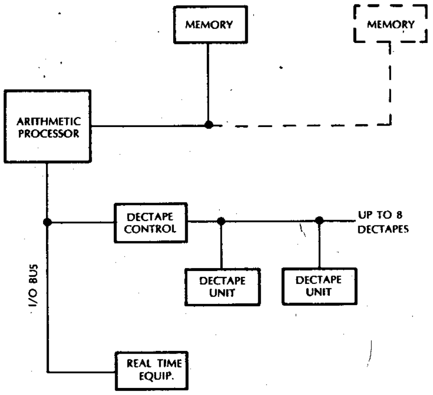

FIGURE 1. SINGLE-USER SYSTEM

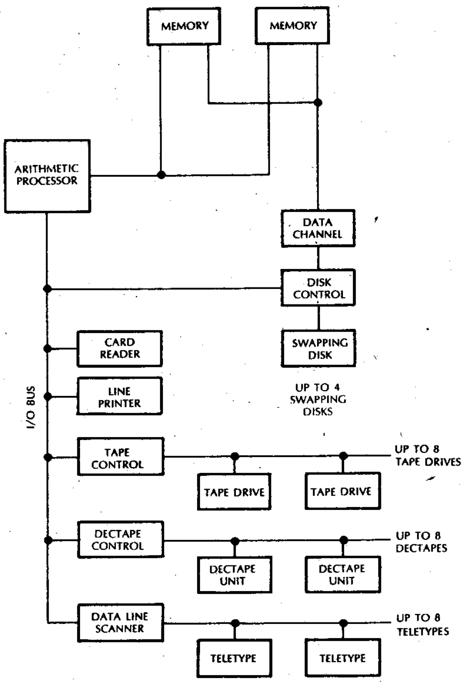

FIGURE 2. 8-USER SWAPPING SYSTEM

The small swapping system shown in Figure 2 can be expanded in eight user groups to the large time-sharing system (Figure 3) which can handle up to 63 users. The large system includes file storage and swapping storage units, additional memory, and more peripheral equipment. For very large systems, a file storage disk may replace or supplement the disk packs. A computer-based communication system may be substituted for the data line scanner, and synchronous data phone units can be used to connect the system to remote batch devices and other computers.

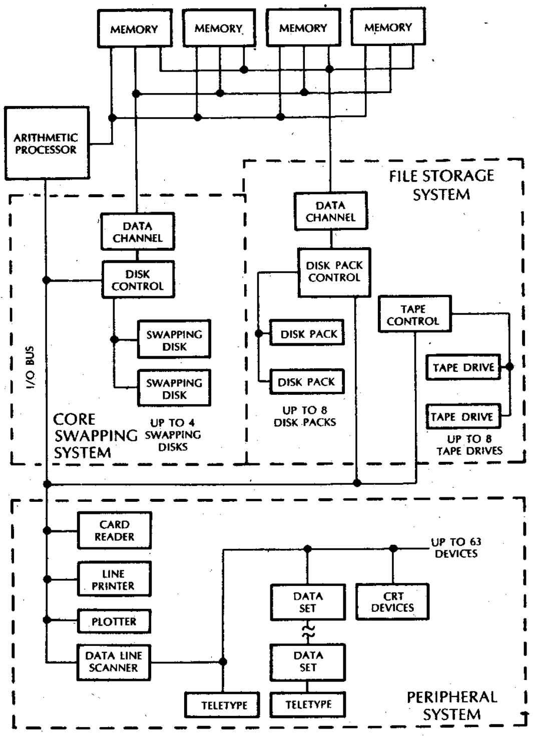

FIGURE 3. LARGE SWAPPING SYSTEM

The dual processor system in Figure 4 is one of many possible multi-processor configurations that the user can tailor his monitor to meet. Since the system shares both peripherals and core memory, both processors can access memory at the same time and can compute in parallel. Such an arrangement doubles the computing power of a single processor and more than doubles cost/effectiveness, since the cost of the additional processor is only a small fraction of overall system cost.

[1] A special random access magnetic tape designed by Digital Equipment Corporation.

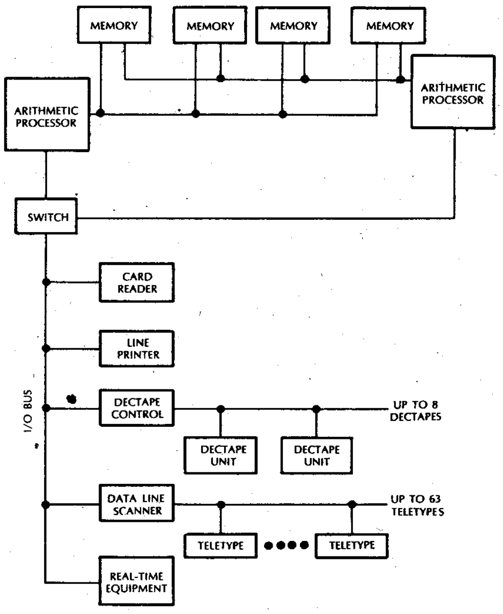

FIGURE 4. DUAL PROCESSOR SYSTEM

In other multi-processor systems, the processors may work independently or communicate through shared memory. One may serve as the input/output processor while the other performs most of the calculations. Or the processors can share all input/output and processing. Multi-processor systems can also combine a PDP-10 arithmetic processor with other DIGITAL computers.

In Summary

The PDP-10 exemplifies the versatility required for today's large computing tasks. With its 366-instruction repertoire, re-entrant software, multi-programming hardware, and flexible priority interrupt system, it provides power and excellent response for a multitude of applications. And with the system's wide range of hardware and software, the user can purchase to serve present needs, yet easily expand to meet future system requirements.

Every PDP-10 is backed by service — software support for a full range of customer assistance, service through a worldwide system of over 60 service centers, and formal training through a variety of available training courses.

At a cost of less than half that of comparable systems, the PDP-10 provides the best price/performance available today — another step toward Digital Equipment Corporation's goal of providing the most for every computing dollar.

INTRODUCTION

SYSTEM DESCRIPTION

DEC PDP-10 software is divided into eight functional groupings with respect to common programming activities, as follows:

a. Source Program Preparation
b. Conversational Language Translators
c. Program Loading and Library Facilities
d. Debugging
e. Utilities
f. Calculators
g. Batch Processing
h. Monitoring

Groups a through g are programs called CUSPs (commonly used system programs) and are run under control of the Single-User, Multiprogramming non-disk, Multiprogramming disk or Swapping Monitor.

Source Program Preparation (EDITOR, LINED, TECO)

The DECtape Editor, LINED (Line Editor for Disk), and the Text Editor and Corrector (TECO) programs can be used to create (and later correct or modify) text files (e.g., Macro-10 and FORTRAN source language programs) for subsequent assembly or compilation. Editor creates and modifies files on DECtape; LINED creates and modifies files on disk; and TECO performs more complex editing functions on any standard I/O devices.

Language Translators (MACRO, F40)

The Macro-10 Assembler (MACRO) and the FORTRAN Compiler (F40) translate source programs written in the Macro-10 and FORTRAN IV languages, respectively, into binary machine language for subsequent loading and execution.

Program Loading and Library Facilities (LOADER, LIB40, JOBDAT, FUDGE2)

Loading is performed by the Linking Loader, which loads specified relocatable binary programs in core, links their references to each other, and searches the appropriate subroutine libraries (e.g., LIB40) for required subroutines. A job data area (JOBDAT) is created by the Loader for each program; this area is used to store the current status of the job during execution. Library files of binary programs can be updated by use of the File Update Generator (FUDGE2).

Debugging (DDT, CREF, GLOB)

After a program is compiled (or assembled), it can be loaded in conjunction with the Dynamic Debugging Technique (DDT) program and debugged. DDT allows the user to control program execution and to modify the program in any of several modes, including symbolic. For purposes of further program analysis (and for documentation), the user can use the Cross Reference Listing (CREF) program, which produces a cross-referenced listing of all symbols within his Macro program, and the Global Cross-Reference Listing (GLOB) program, which produces one to three listings of all global symbols encountered in one or more programs.

Utilities (PIP, CODE, SRCCOM, BINCOM)

A variety of utility programs are available for general-purpose data handling. Among these programs are: the Peripheral Interchange Program (PIP), which transfers data between any standard I/O devices; Code Translator (CODE), which performs translations between standard ASCII codes and code of other manufacturers; Source Compare (SRCCOM), which compares two versions of an ASCII file; and Binary Compare (BINCOM), which compares two versions of a binary file.

Conversational Languages (AID, BASIC)

Two problem-solving conversational languages for scientists, engineers, and students are included as part of the PDP-10 software: the Algebraic Interpretive Dialogue (AID), a program based on the RAND JOSS[TM]algebraic language; and Advanced BASIC[®] , a conversational language for scientific, business, and educational applications that includes among many other features a special set of matrix processing operations.

[TM]JOSS is the trademark and service mark of the RAND Corporation for its computer program and services using that program.
[®] Registered, Trustees of Dartmouth College.

Batch Processing (BATCH, STACK)

The Batch Processor (BATCH) monitors the sequential execution of a series of user jobs with a minimum of operator attention; operates as one of the "users" in a time-sharing environment and runs concurrently with the Batch-controlled jobs (as well as other jobs on the system); and permits constant communication by the operator. Job Stacker (STACK) prepares input stacks for BATCH and processes output stacks from BATCH.

Monitors

PDP-10 software includes two major categories of Monitors: the Single-User Monitor (10/30 configuration) and the Time-Sharing Monitors (10/40 and 10/50 configurations). The latter category includes th Multiprogramming non-disk Monitor (10/40), Multiprogramming disk Monitor (10/40) and the Swapping Monitor (10/50). The Swapping Monitor was used in the generation of all examples in this manual.

The 10/30 Monitor is a subset of the 10/40 and 10/50 Monitor. They are compatible at the source and relocatable binary levels. The 10/40 and 10/50 Monitors are compatible at the source, relocatable binary, and saved core image levels.

SYSTEM OPERATION

The following basic procedures and rules are necessary to communicate with the Monitor and load and execute DEC commonly used system programs (CUSPs), as well as user programs.

Step Procedure

1. To establish communication with the Monitor, place the Teletype in Monitor Command Mode by typing ↑C (i.e., hold down the CTRL key while striking C; Monitor will respond with a period (.). If you have a 10/30 or a 10/40 system, skip the LOGIN procedure in the next step.

2. Type LOGIN followed by carriage return. The system will respond with

   JOB n NAME OF SYSTEM

   followed by a number sign. Then type your project-programmer number, followed by a carriage return. The system will then type

   PASSWORD:

   Then type your secret password followed by a carriage return. The system will keep it secret by not printing it on the paper. If the project-programmer number agrees with the password, you will be logged, and any messages of the day will be typed for you.

3. Then, direct Monitor to load and start a program from the System Library (.R prog), start a program already loaded in core (.START), discontinue your job (.KJOB), or perform any of a variety of other operations. A complete list of Time-Sharing Monitor commands is given in Table 9-1.

All CUSPs (except EDITOR and LINED) supplied by DEC are device independent; therefore, the user must tell the CUSP, via a command string typein, which devices to use. Readiness to receive a command string is signalled by the CUSP via an asterisk (*) typeout after loading. For example, when the FORTRAN IV Compiler has been called and it has responded with an asterisk, the user types in a command string indicating:

a. The device containing the source program to be compiled
b. The device on which the binary output is to be placed and
c. The device on which the compilation listing is to be written

```
*binary-output-device, listing-device ← source-device
```

Devices are specified by a 3-character device name (a fourth character, a digit, specifies the particular unit in the case of DECtapes, teletypes, and magnetic tapes), followed by a colon.

Device	Device Name
Card reader	CDR:
Card punch	CDP:
Line printer	LPT:
Paper tape reader	PTR:
Paper tape punch	PTP:
Teletype	TTY: or TTYn:
DECtape	DTAn:
Magnetic tape	MTAn:
Disk	DSK:

For file-oriented devices (DECtape and disk), a filename (maximum of six characters) is also required following the device name to specify either the specific file to be read or the filename to be assigned to the output. A filename can be further specialized by adding a 3-character extension name to it, preceded by a period (.). Extension names are generally used to classify a file into a particular category, and certain standard extensions are used and recognized throughout the system (e.g., .REL for relocatable binary files, .SAV for saved core image files, .MAC for Macro-10 source files, .F4 for FORTRAN source files, etc.). The following example shows a sample FORTRAN command string.

Example:

DTA1:BIN.REL,LPT: ← DTA0:SOURCE	Compile the file designated as SOURCE on DECtape 0; write the binary output on DECtape 1, designating it BIN.REL; print the listing on the line printer.

Batch Processing (BATCH, STACK)

The Batch Processor (BATCH) monitors the sequential execution of a series of user jobs with a minimum of operator attention; operates as one of the "users" in a time-sharing environment and runs concurrently with the Batch-controlled jobs (as well as other jobs on the system); and permits constant communication by the operator. Job Stacker (STACK) prepares input stacks for BATCH and processes output stacks from BATCH.

Monitors

PDP-10 software includes two major categories of Monitors: the Single-User Monitor (10/30 configuration) and the Time-Sharing Monitors (10/40 and 10/50 configurations). The latter category includes th Multiprogramming non-disk Monitor (10/40), Multiprogramming disk Monitor (10/40) and the Swapping Monitor (10/50). The Swapping Monitor was used in the generation of all examples in this manual.

The 10/30 Monitor is a subset of the 10/40 and 10/50 Monitor. They are compatible at the source and relocatable binary levels. The 10/40 and 10/50 Monitors are compatible at the source, relocatable binary, and saved core image levels.

SYSTEM OPERATION

The following basic procedures and rules are necessary to communicate with the Monitor and load and execute DEC commonly used system programs (CUSPs), as well as user programs.

<u>Step</u> <u>Procedure</u>

1. To establish communication with the Monitor, place the Teletype in Monitor Command Mode by typing ↑C (i.e., hold down the CTRL key while striking C; Monitor will respond with a period (.). If you have a 10/30 or a 10/40 system, skip the LOGIN procedure in the next step.

2. Type LOGIN followed by carriage return. The system will respond with

   JOB n NAME OF SYSTEM

   followed by a number sign. Then type your project-programmer number, followed by a carriage return. The system will then type

   PASSWORD:

   Then type your secret password followed by a carriage return. The system will keep it secret by not printing it on the paper. If the project-programmer number agrees with the password, you will be logged, and any messages of the day will be typed for you.

3. Then, direct Monitor to load and start a program from the System Library (.R prog), start a program already loaded in core (.START), discontinue your job (.KJOB), or perform any of a variety of other operations. A complete list of Time-Sharing Monitor commands is given in Table 9-1.

TYPOGRAPHIC CONVENTIONS IN THIS MANUAL

All computer typeouts are underscored (single line) or enclosed in brackets (multiple lines).

All operator typeins are not underscored.

SYMBOLOGY USED IN CONSOLE EXAMPLES

↑C	Hold down the CTRL key while striking C. Normally echoes as ↑C.
↑x	Hold down the CTRL key while striking the "x" key, where "x" is any character. Normally echoes as ↑x.

Some special control symbols and their respective key designations for Models 33, 35, and 37 Teletypes are given below.

Key Designation	Symbol in This Manual	Models 33 and 35	Model 37
TAPE	↑R	Hold down CTRL key while striking R.	Same as 33/35
~~TAPE~~ (not-TAPE)	↑T	Hold down CTRL key while striking T.	Same as 33/35
BELL	↑G	Hold down CTRL key while striking G.	Same as 33/35
TAB (horizontal tab)	⇥ or ↑I	Hold down CTRL key while striking I.	Strike TAB key
FORM	↑L	Hold down CTRL key while striking L.	Same as 33/35
VT (vertical tab)	↑K	Hold down CTRL key while striking K.	Same as 33/35
XON	↑Q	(Initialize paper tape reader input.) Hold down CTRL key while striking Q.	Same as 33/35
XOFF	↑S	(Terminate paper tape reader input.) Hold down CTRL key while striking S.	Same as 33/35
←	←	Hold down the SHIFT key while striking O.	Strike ← key
RETURN	↵	Strike the RETURN key. Normally echoes back as a carriage return, line feed.	Same as 33/35
ALTMODE or PREFIX or ESC	$	Strike the ESC key (sometimes labeled ALTMODE or PREFIX)	Same as 33/35
[	[	Hold down the SHIFT key while striking K.	Strike [key

Key Designation	Symbol in This Manual	Models 33 and 35	Model 37
]	]	Hold down the SHIFT key while striking M.	Strike] key
↑	↑	When appearing alone (as in DDT), hold down the SHIFT key while striking N.	Strike ↑ key
<	<	Hold down the SHIFT key while striking ",".	Strike < key
>	>	Hold down the SHIFT key while striking ".".	Strike > key
LINE FEED	↓	Strike the LINE-FEED key.	Strike LINE SPACE key
RUBOUT	RUBOUT	Strike the RUBOUT key. Normally echoes back as a backslash (\), XXX, or a repeat of the character erased.	Strike DELETE key
\	\	Hold down the SHIFT key while striking L.	Strike the \ key
SPACE BAR	␣	Strike the space bar to space to indicated position. TAB can also be used in most instances.	Same as 33/35

NOTE

Due to recent changes in ASCII, some terminals may have the keytops "∧ " (caret) and "___" (underline); these characters have the same codes as "↑" and "←", respectively. Where possible, DEC will supply all teletypes with the arrow characters.

DEMONSTRATION PROGRAMS

The following demonstration programs illustrate the simplicity and flexibility of a PDP-10 software system:

a. Demonstration #1 is a typical example of the procedure for creating a FORTRAN main program source file and a Macro-10 subprogram file. These two files are then translated, loaded, and executed together. A bug is encountered during execution, and the DDT (Dynamic Debugging Technique) program is used to correct the erroneous instruction. The programs are now executed successfully, their core image is saved for future execution, and the original source file is altered to reflect the correction made to the binary code.

b. Demonstration #2 is a more complex example. The sequence of operations is similar to that of Demonstration #1 (source program file creation, translation, loading, execution, debugging, saving the core image, and altering the source code to reflect changes made to the object code). In addition, such procedures as leaving the current job, logging in and beginning a second job, and then later returning to the original job are included. Figure 1-1 contains the flow diagram for Demonstration #2.

Demonstration #1

```
↑C
.LOGIN
JOB 10    4S.51G DEC PDP-10 #40
#10,63
PASSWORD:
1641    01-JUL-69      TTY22
THE DISK WILL BE REFRESHED AT 1300 HOURS DAILY

↑C
.ASSIGN DTA
DTA2 ASSIGNED


.MAKE MAINPG

*I C     FORTRAN PROGRAM FOR TYPING TTY PHYSICAL NAME
         CALL GETTYN(R)
         TYPE 6,R
6        FORMAT(' THE NAME OF YOUR TELETYPE IS:   ',A5)
         END
$$

*EX$$

EXIT
↑C


.MAKE SUBRTE.MAC

*ITITLE GETTYN MACRO SUBROUTINE
SUBTTL SUBROUTINE TO GET TTY NAME AND CONVERT TO ASCII.
INTERNAL GETTYN
         AC4=4
         AC5=5
         AC6=6
GETTYN:  0
         CALL AC4,[SIXBIT/GETLIN/]         ;GET TTY NAME.
GETBYT:  ILDB AC6,NMPTR1          ;GET A SIXBIT CHARACTER.
         SKIPN AC6                ;DONE?
         JRST NAMDON              ;YES.
         ADDI AC6,40              ;NO, CONVERT CHAR TO ASCII.
         IDPB AC6,NMPTR2          ;SAVE CONVERTED CHARACTER.
         JRST GETBYT              ;GO GET NEXT CHARACTER.
NAMDON:  MOVE AC5,@(16)           ;STORE NAME.
         JRA 16,1(16)             ;RETURN TO MAINPG.
NMPTR1:  POINT 6,AC4-1,35
NMPTR2:  POINT 7,AC5-1,34
         END
$$
*EX$$

EXIT
↑C
```

Log into the system by typing LOGIN, followed by the prescribed "login" information for your particular system. The monitor responds with time, date, and Teletype number.

ASSIGN DTA assigns a DECtape for storage of the completed program.

MAKE MAINPG calls in TECO (Text Editor and Corrector program) to create MAINPG, your FORTRAN IV source program file. The text of the source program is preceded by TECO insert command I. To terminate the text, type two ALTMODEs $$. TECO command EX$$ deposits the file in your disk area and returns you to the monitor.EXIT. ↑ C acknowledges the return to the monitor.

MAKE SUBRTE.MAC creates a MACRO-10 source program file SUBRTE.MAC. This subroutine with the program name of GETTYN is called from the above FORTRAN program to obtain the Teletype name in SIX-BIT code and convert it to USASCII code for outputting. The I, $$, and EX$$ commands perform the functions previously mentioned.

```
.EXECUTE SUBRTE/CREF,MAINPG/L
FORTRAN:  MAINPG
MACRO: GETTYN
LOADING

LOADER 5K CORE



THE NAME OF YOUR TELETYPE IS:
EXIT
↑C

.DIRECT


DIRECTORY 27,20 Ø9Ø1      29-JAN-69

ØØ3EDS.TMP        Ø1      29-JAN-69
MAINPG            Ø1      29-JAN-69
SUBRTE.MAC        Ø1      29-JAN-69
ØØ3LOA.TMP        Ø1      29-JAN-69
ØØ3MAC.TMP        Ø1      29-JAN-69
ØØ3SVC.TMP        Ø1      29-JAN-69
ØØ3CRE.TMP        Ø1      29-JAN-69
MAINPG.LST        Ø1      29-JAN-69
MAINPG.REL        Ø1      29-JAN-69
SUBRTE.REL        Ø1      29-JAN-69
SUBRTE.LST        Ø3      29-JAN-69
ØØ3PIP.TMP        Ø1      29-JAN-69

TOTAL BLOCKS      14

EXIT
↑C


.LIST MAINPG.LST

EXIT
↑C

.CREF

EXIT
↑C

.DEBUG SUBRTE,MAINPG

LOADING

LOADER 7K CORE
```

EXECUTE SUBRTE/CREF,MAINPG/L instructs the system to: 1. Assemble SUBRTE.MAC, generating a cross reference listing file (CREF), and compile MAINPG, creating a normal listing file (L). 2. Load the two resulting relocatable binary files together. 3. Start execution. System acknowledges each step as it is being executed and types out the total core requirement for loading.

The program has a bug. It didn't complete the message: THE NAME OF YOUR TELETYPE IS:

To find the bug, it may be helpful to list the directory of your disk area by using the command DIRECT. Besides the two text files created with TECO, there are many others. The REL files contain the relocatable binary output from the assembly and compilation. The LST files contain the listings. The TMP files are temporary command files created by the COMPIL CUSP. These include 003CRE.TMP, which contains the names of LST files to be output to the line printer when a CREF command is given.

You can now print the listing files for examination. LIST MAINPG.LST outputs the listing file produced when the FORTRAN program was compiled. CREF outputs the CREF listing file produced when the MACRO-10 subroutine was assembled. Examination shows that the MOVE instruction in the MACRO-10 subroutine should be a MOVEM.

To debug your program, load the DDT (Dynamic Debugging Technique) program by typing DEBUG SUBRTE,MAINPG. Note that assembly and compilation are not repeated since the REL files are more recent than the source files.

```
GETTYN$:          NAMDON/ MOVE AC5,@Ø(16) MOVEM AC5,@(16)
$G

THE NAME OF YOUR TELETYPE IS:  TTY13
EXIT
↑C


.SAVE DTA2:IMAGE
JOB SAVED
↑C

.TECO SUBRTE.MAC

* BJNMOVE$IM$ØLT$$
NAMDON: MOVEM AC5,@(16)          ;STORE NAME.
*EX$$

EXIT
↑C
```

GETTYN$: accesses the symbol table for the MACRO subroutine, and NAMDON/ accesses the location of the erroneous instruction. Type in the correction MOVEM AC5, @ (16) and use $G to re-execute the program so that you can check the result. The bug is out! The message is completed: THE NAME OF YOUR TELETYPE IS: TTY13.

The final steps are to store the core image of your program on DECtape and correct the source file. SAVE DTA2:IMAGE stores the program on DECtape 2, giving it the name IMAGE. TECO SUBRTE.MAC calls in TECO to correct the source file. And the TECO command string BJNMOVEIM0LT$$ searches for the word MOVE, inserts an M after it, and types out the corrected line. EX$$ deposits the corrected source program in your disk area and returns to the monitor. The user can now log off the system or start some other program.

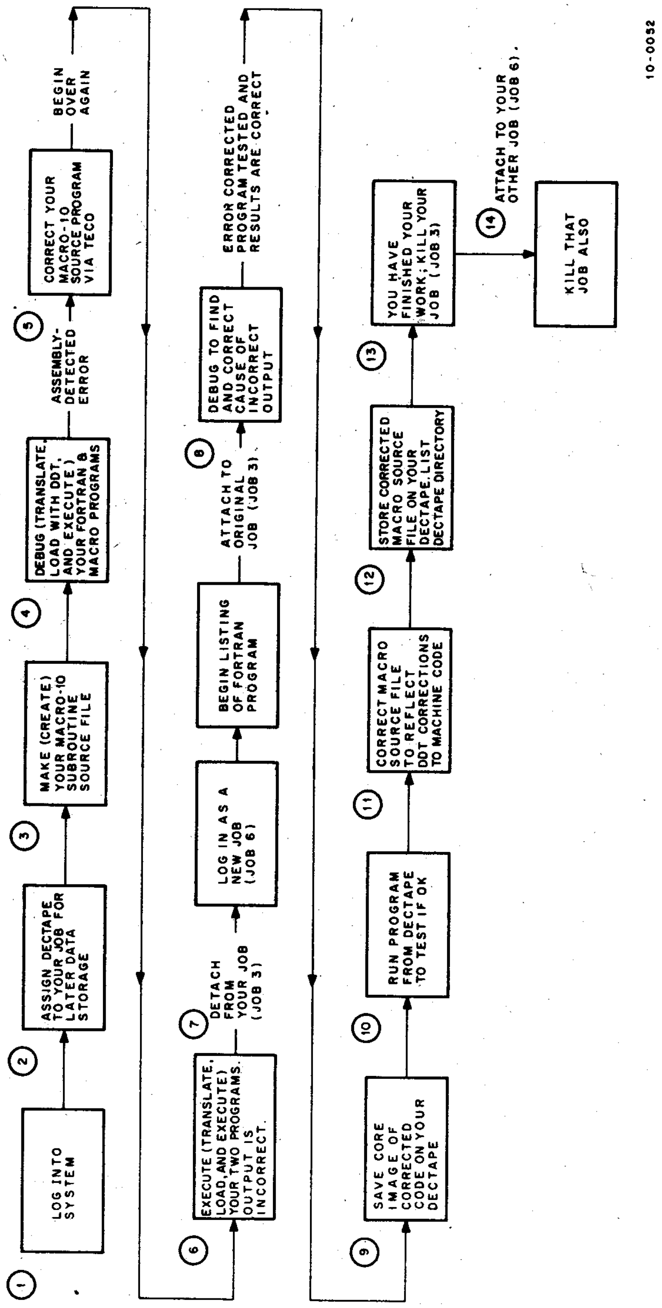
1
LOG INTO SYSTEM
2
ASSIGN DECTAPE TO YOUR JOB FOR LATER DATA STORAGE
3
MAKE (CREATE) YOUR MACRO-10 SUBROUTINE SOURCE FILE
4
DEBUG (TRANSLATE, LOAD WITH DDT, AND EXECUTE) YOUR FORTRAN & MACRO PROGRAMS
ASSEMBLY-DETECTED ERROR
5
CORRECT YOUR MACRO-10 SOURCE PROGRAM VIA TECO
BEGIN OVER AGAIN
6
EXECUTE (TRANSLATE, LOAD, AND EXECUTE) YOUR TWO PROGRAMS. OUTPUT IS INCORRECT.
7
DETACH FROM YOUR JOB (JOB 3)
LOG IN AS A NEW JOB (JOB 6)
BEGIN LISTING OF FORTRAN PROGRAM
ATTACH TO ORIGINAL JOB (JOB 3)
8
DEBUG TO FIND AND CORRECT CAUSE OF INCORRECT OUTPUT
ERROR CORRECTED PROGRAM TESTED AND RESULTS ARE CORRECT
9
SAVE CORE IMAGE OF CORRECTED CODE ON YOUR DECTAPE
10
RUN PROGRAM FROM DECTAPE TO TEST IF OK
11
CORRECT MACRO SOURCE FILE TO REFLECT DDT CORRECTIONS TO MACHINE CODE
12
STORE CORRECTED MACRO SOURCE FILE ON YOUR DECTAPE, LIST DECTAPE DIRECTORY
13
YOU HAVE FINISHED YOUR WORK; KILL YOUR JOB (JOB 3)
14
ATTACH TO YOUR OTHER JOB (JOB 6).
KILL THAT JOB ALSO
10-0052

Figure 1-1 Demonstration Program #2 Flow Diagram

Demonstration #2

```
 .LOGIN
  JOB 3        4S34
  #27,2Ø
  Ø955     27-JAN-69     TTY13
  THE DISK WILL BE REFRESHED DAILY AT 4:45 PM UNTIL FURTHER NOTICE.

  ↑C




 .ASSIGN DTA DT
 DTA2 ASSIGNED




.MAKE RANDOM

*ITITLE RANDOM NUMEEBER GENERATING SUBROUTINE
SUBTTL CHARLIE PROGRAMMER      27 JAN 1969
;RANDOM NUMBER GENERATING SUBROUTINE
$$
*I;THE FORTRAN CALLING SEQUENCE IS --
;       CALL RANDOM (ARG)
;WHERE ARG SPECIFIES THE LOCATION AT WHICH THE RESULTING
;SINGLE PRECISION FLOATING POINT RANDOM NUMBER WILL BE
;STORED.  NUMBERS PRODUCED BY THIS ROUTINE ARE PSEUDO:RANDOM
;NUMBERS BUT ARE UNIFORMLY DISTRIBUTED OVER [Ø,1].
$$
*INTERNAL RANDOM
        ACX=5
        ACY=6
        ACZ=ACY+1  ;ACCUMULATOR SYMBOLIC DEFINITIONS.
$$
*IRANDOM:  Ø     ;ENTERED BY JSA 16,RANDOM.
           CALL ACX,[SIXBIT/TIMER]  ;GET TIME IN CLOCK TICKS.
           ANDI ACX,3  ;USE TIME TO SELECT 1-4 ITERATIONS.
$$
*IRLOOP:   MOVE ACY,RNUMBR   ;FETCH PREVIOUS PSEUDO-RANDOM NUMBER.
           MUL ACY,MAGIC     ;MULTIPLICATIVE RANDOM NUMBER GENERATOR.
           MOVEM ACZ,RNUMBR  ;SAVE NEXT PSEUDO-RANDOM NUMBER.
           SOJGE ACX,RLOOP   ;ITERATE AGAIN?
$$
*I         LSH ACZ,-↑D8      ;CONVERT TO FLOATING POINT FORMAT.
           TLO ACZ,2ØØØØ     ;IN THE RANGE [Ø,1].
           FADRI ACZ,Ø       ;NORMALIZE.
           MOVEM ACZ,@(16)   ;STORE RESULT, AND
           JRA 16,1(16)      ;**RETURN**.
```

Log into the system by typing LOGIN (may be abbreviated as LOG); Monitor responds with the job number assigned to your job and the version number of the Monitor. Following the typeout of the # symbol, type in your project-programmer number. Monitor then waits for a password. Type your password (echo-typeout is suppressed). If your password matches correctly with your project-programmer number, Monitor types out the correct time, date, and the physical name of the teletype you are using. Monitor may type out some informative messages and return you to Monitor level. At this point, any Monitor command can be typed.

Assign a DECtape unit to the job for later storage of files, and assign it the logical name DT. Monitor responds that DECtape unit 2 has been assigned to the job. Mount an available reel on this unit, and place the WRITE switch in the WRITE-ENABLED position. The reel contains the FORTRAN source program for later use. From this point, refer to the DECtape unit as either DTA2: or DT:.

Now, create the source program file for the Macro-10 subroutine to be run in conjunction with the FORTRAN program. TECO (Text Editor and Corrector) can be used to create such a text file. Type MAKE RANDOM to call in the TECO program and cause TECO to open a file for creation; give it the filename RANDOM. After TECO has responded with an asterisk, type an Insert command (I) followed by the first portion of the text of your Macro-10 source program. Note that a typing mistake was made on the first line - NUME; this can be corrected by pressing the RUBOUT key to echo the previous character and then typing the correct character. If a typing error occurs several characters back, press the RUBOUT key repeatedly until you have reached the erroneous character. To avoid overflowing the input command buffer, break the text into several segments, rather than typing it as one continuous block. This is done by typing two successive ALTMODEs (an ALTMODE echoes as $) after every six or seven lines of text to cause the contents of the input command buffer to be transferred to the TECO output buffer and the input command buffer to be cleared. Following the subsequent asterisk typeout, repeat the Insert command before continuing the text. After typing the program, request a typeout of the entire text by typing HT$$. Notice that an Insert command was not typed at the beginning of the third segment of the text. Luckily, TECO took the "I" in INTERNAL as the Insert command, but this resulted in NTERNAL. Correct this by typing BJ (set the TECO pointer at the beginning of the text), S (Search) for NTERNAL, 7R (Reverse the pointer seven characters), II (Insert an "I"), and 0LT (Type out the corrected Line). Note that each command step is terminated by an ALTMODE ($). Also, insert a space following PP in the line ;ACM.......PP83-89) and request that the corrected line be typed out. Type EX $$ to direct TECO to write out its output buffer onto disk, assign it the filename previously specified in the MAKE command, and close the file.

```
$$
*I;THE MULTIPLIER USED IS 5↑15 (SEE COMPUTER REVIEWS, VOL 6, #3,
;REVIEW NUMER 7725, AND THE REFERENCED PAPER IN JOURNAL OF
;ACM, JANUARY, 1965, PP83-89).
MAGIC:      5*5*5*5*5*5*5*5*5*5*5*5*5*5*5 ;THE MULTIPLIER.
RNUMBR:     1                 ;THE NEXT RANDOM NUMBER IS ALWAYS HERE.
;THE ITERATION STARTS FROM A VALUE OF 1.
PATCH:      BLOCK 10          ;PATCHING SPACE.
            END
$$
*HT
$$
TITLE RANDOM NUMBER GENERATING SUBROUTINE
SUBTTL CHARLIE PROGRAMMER      27 JAN 1969
;RANDOM NUMBER GENERATING SUBROUTINE
;THE FORTRAN CALLING SEQUENCE IS --
;        CALL RANDOM (ARG)
;WHERE ARG SPECIFIES THE LOCATION AT WHICH THE RESULTING
;SINGLE PRECISION FLOATING POINT RANDOM NUMBER WILL BE
;STORED.  NUMBERS PRODUCED BY THIS ROUTINE ARE PSEUDO:RANDOM
;NUMBERS BUT ARE UNIFORMLY DISTRIBUTED OVER [0,1].
NTERNAL RANDOM
        ACX=5
        ACY=6
        ACZ=ACY+1  ;ACCUMULATOR SYMBOLIC DEFINITIONS.
RANDOM:  0      ;ENTERED BY JSA 16,RANDOM.
            CALL ACX,[SIXBIT/TIMER]  ;GET TIME IN CLOCK TICKS.
            ANDI ACX,3  ;USE TIME TO SELECT 1-4 ITERATIONS.
RLOOP:    MOVE ACY,RNUMBR  ;FETCH PREVIOUS PSEUDO-RANDOM NUMBER.
            MUL ACY,MAGIC     ;MULTIPLICATIVE RANDOM NUMBER GENERATOR.
            MOVEM ACZ,RNUMBR ;SAVE NEXT PSEUDO-RANDOM NUMBER.
            SOJGE ACX,RLOOP  ;ITERATE AGAIN?
          LSH ACZ,-↑D8      ;CONVERT TO FLOATING POINT FORMAT.
            TLO ACZ,20000     ;IN THE RANGE [0,1].
            FADRI ACZ,0       ;NORMALIZE.
            MOVEM ACZ,@(16)   ;STORE RESULT, AND
            JRA 16,1(16)      ;**RETURN**.
;THE MULTIPLIER USED IS 5↑15  (SEE COMPUTER REVIEWS, VOL 6, #3,
;REVIEW NUMBER 7725, AND THE REFERENCED PAPER IN JOURNAL OF
;ACM, JANUARY, 1965, PP83-89).
MAGIC:      5*5*5*5*5*5*5*5*5*5*5*5*5*5*5  ;THE MULTIPLIER.
RNUMBR:     1                 ;THE NEXT RANDOM NUMBER IS ALWAYS HERE.
;THE ITERATION STARTS FROM A VALUE OF 1.
PATCH:      BLOCK 10          ;PATCHING SPACE.
            END  $
*BJSNTERNAL$7RIIS0LT$$
INTERNAL RANDOM
*SPP83$-2CI $0LT$$
;ACM, JANUARY, 1965, PP 83-89).
*EXIT$$

EXIT
↑C
```

Demonstration Program #2 Continues On Next Page

```
.DIRECT

DIRECTORY 27,2Ø 1Ø19    27-JAN-69

ØØ3EDS.TMP      Ø1      27-JAN-69
RANDOM          Ø3      27-JAN-69
ØØ3PIP.TMP      Ø1      27-JAN-69

TOTAL BLOCKS    Ø5

EXIT
↑C
```

```
.RENAME RANDOM.MAC=RANDOM

FILES RENAMED
RANDOM

EXIT ↑C
```

```
.DEBUG RANDOM/CREF,DT:ARRIVE/L
FORTRAN:  ARRIVE.F4
MACRO: RANDOM
A       ØØØØØ1  Ø4Ø24Ø  ØØØØ26'     CALL ACX,[SIXBIT/TIMER]  ;GE
T TIME IN CLOCK TICKS.

?1 ERROR DETECTED
LOADING      /
?EXECUTION DELETED
LOADER 8K CORE
EXIT
↑C
```

```
.TECO RANDOM.MAC

[3 K CORE]
*BJN/TIMER$I/$ØLT$$
           CALL ACX,[SIXBIT/TIMER/]  ;GET TIME IN CLOCK TICKS.
*EXIT$$

EXIT
↑C
```

Typing DIRECT causes the directory of the disk area to be typed out on the console. In addition to the RANDOM file, there are two other files, 003EDS.TMP and 003PIP.TMP. These files are temporary command files created by the COMPIL CUSP and contain the commands generated by MAKE and DIRECT, respectively. Note that the assigned job number forms the first three characters of the filenames.

Change the name of the text file from RANDOM to RANDOM.MAC by typing

RENAME RANDOM.MAC=RANDOM.

Standard filename extensions should be used (e.g., .MAC for Macro-10 source program files, .F4 for FORTRAN source program files, .REL for relocatable binary files, etc.).

Use the DEBUG command as follows:

a. Assemble your Macro-10 source program, RANDOM. Its filename extension of .MAC identifies it as a Macro program. Request that a CREF (cross-reference) listing file be produced. At a later time, this file can be listed on the printer via the CREF command.

b. Compile your FORTRAN source program, ARRIVE, which has a filename extension of .F4, identifying it as a FORTRAN program. Request a normal listing file (/L). This previously prepared program file is on the DECtape reel that was mounted on DECtape 2 (symbolic name DT:).

c. Load the two relocatable binary files produced by the assembly and compilation processes and also load the Dynamic Debugging Technique (DDT) program to examine and debug the program coding.

d. If no major errors were encountered during translation and loading, begin execution under control of DDT.

In this case, however, a source coding error was encountered in the Macro-10 source program (a slash was not typed following TIMER) and execution of the programs is inhibited.

Type TECO RANDOM.MAC to recall TECO and to open an already existing file, RANDOM.MAC, for editing. Type the command string shown to insert a slash after TIMER . Set the TECO pointer at the beginning (BJ) of the text, do a nonstop (N) search for /TIMER, insert a slash (I/) following it, and type the current line (0LT).

```
.DELETE *.REL,*.LST

FILES DELETED
ARRIVE  REL
RANDOM  REL
ARRIVE  LST
RANDOM  LST

EXIT
↑C
```

```
.EXECUTE RANDOM/CREF,DT:ARRIVE/L
FORTRAN:  ARRIVE.F4
MACRO: RANDOM
LOADING

LOADER 6K CORE



RANDOM INTERARRIVAL TIME GENERATOR FOR POISSON PROCESSES

TYPE MEAN WAITING TIME PLEASE:     1ØØ

TYPE NUMBER OF SAMPLE TIMES DESIRED:1Ø

T =  Ø.77751Ø67E+Ø4
T =  Ø.79615811E+Ø4
T =  Ø.79469139E+Ø4
T =  Ø.78677823E+Ø4
T =  Ø.781Ø3187E+Ø4
T =  Ø.777ØØ529E+Ø4
T =  Ø.7782Ø5Ø1E+Ø4
T =  Ø.7772547ØE+Ø4
T =  Ø.7953Ø156E+Ø4
T =  Ø.779176Ø9E+Ø4

TYPE MEAN WAITING TIME PLEASE:    ↑C
```

```
.DETACH
```

```
.LOGIN
JOB 6   4S34
#27,2Ø
1Ø29    27-JAN-69    TTY13
THE DISK WILL BE REFRESHED DAILY AT 4:45 PM UNTIL FURTHER NOTICE.

↑C
```

To repeat the translation of the programs, delete those files from your disk area that were created by the DEBUG process. These files are the two relocatable binary files (these were automatically given a filename extension of .REL) and the two listing files (these were automatically given a filename extension of .LST). The form *.ext refers to all files, regardless of filename, that have the specified extension. In this example, DELETE *.REL, *.LST causes all files with an extension of .REL and all files with an extension of .LST to be deleted. To conserve disk space, note that all temporary command files generated by Monitor commands can be periodically deleted by typing DELETE *.TMP.

An attempt is made to translate, load, and execute without DDT. The EXECUTE command has the same general format as the DEBUG command. After loading, the program will automatically begin execution.

Translation was successful; the programs are loaded; and execution is begun.

However, there is an error. The output is far from random and conspicuously in the wrong range. Return to Monitor level by typing ↑C.

The following is an example of how to detach from the current job and begin a second job without affecting the status of the current job. Detach the Teletype console by typing DETACH. A new job can now be initiated. The current job (job #3) remains in its present status until you attach to it again.

Type LOGIN (or LOG) to request another job number. Job #6 is assigned. Perform the same procedures for logging in as in Step 1.

```
.LIST ARRIVE.LST

↑C

.CCONT

2K CORE



.ATTACH 3 [27,20]
.


.DEBUG RANDOM,ARRI
LOADING

LOADER 8K CORE



$G

RANDOM INTERARRIVAL TIME GENERATOR FOR POISSON PROCESSES

TYPE MEAN WAITING TIME PLEASE:    100

TYPE NUMBER OF SAMPLE TIMES DESIRED:10

T =  0.77751067E+04
T =  0.79615811E+04
T =  0.79469139E+04
T =  0.78677823E+04
T =  0.78103187E+04
T =  0.77700529E+04
T =  0.77725470E+04
T =  0.80251919E+04
T =  0.78686508E+04
T =  0.77917609E+04


TYPE MEAN WAITING TIME PLEASE:    ↑C

.DDT
```

LIST the listing file generated by the compilation of the FORTRAN program on the line printer.

Once the listing has begun, interrupt it by typing ↑C to return to Monitor level.

Type CCONT to continue the listing and maintain the console at Monitor level.

Now, detach from this job and reattach to the original job by typing ATTACH job# [project, programmer]. (ATTACH can be abbreviated to AT.)

Now, attached to the original job (job #3) other tasks can be performed while the listing is being completed.

The error that caused the incorrect results (a 0 was omitted in the TLO ACZ, 20000 instruction) is determined by scanning the teletype sheet. Now, DEBUG the programs to correct this error.

The DEBUG process finds that two relocatable binary files (created during the previous EXECUTE process) are more recent than their related source files. Therefore, no retranslation is needed, and the existing .REL files are immediately loaded. Execution is begun under control of DDT, and DDT awaits a command typein.

A $G (ALTMODE G) transfers control to the programs and begins execution. Again, incorrect answers result.

To return to the DDT program, type (↑C) to return to Monitor level; then, type "DDT."

```
RANDOM$:        RLOOP+5/        TLO ACZ,20000    TLO ACZ,200000
=661340,,200000 ←TLO ACZ,200000
```

```
MAIN.$: 12P+10$T/       RED:'   "/RED: /
12P+10$T/       RED:  LINEFEED
12P+11/ $)       "/ '$)/
12P+11$T/         '$)
```

```
$G

RANDOM INTERARRIVAL TIME GENERATOR FOR POISSON PROCESSES

TYPE MEAN WAITING TIME PLEASE:    100

TYPE NUMBER OF SAMPLE TIMES DESIRED:  10

T =  0.13360079E+03
T =  0.59776286E+02
T =  0.31049938E+02
T =  0.43099106E+02
T =  0.12045378E+02
T =  0.20455130E+02
T =  0.21030612E+02
T =  0.39184699E+01
T =  0.39241011E+02
T =  0.45517379E+02


TYPE MEAN WAITING TIME PLEASE:    ↑C
```

Open the DDT symbol table for the Macro program by typing RANDOM$: (the program name, taken from the TITLE statement of your source program, followed by an ALTMODE and a colon). Now, any of the symbolic tags contained in this program can be used.

Open the location containing the erroneous instruction by typing the address of the location, relative to a symbolic tag (in this case, RLOOP+5). DDT types out the contents of this location in symbolic form. Now, type in the correct contents, also in symbolic form.

Typing an equal (=) sign causes the new contents to be typed out in halfword mode. Typing a left arrow causes the contents to be typed in symbolic. The proper instruction has been entered correctly.

During execution, no spacing was performed following the second request for input. To correct this condition, open the symbol table for the FORTRAN program (FORTRAN programs are assigned the program name MAIN. unless otherwise titled by the programmer), and correct that portion of the literal stored in locations 12P+10$T and 12P+11$T by inserting a space after the colon (DESIRED:). Examine the two locations to ensure that the correction was made properly (a LINEFEED causes the next sequential location to be opened).

Type $G to restart execution.

The results seem to be correct. Return to Monitor level.

```
.SAVE DT:PROGA
JOB SAVED
↑C

.RUN DT:PROGA

RANDOM INTERARRIVAL TIME GENERATOR FOR POISSON PROCESSES

TYPE MEAN WAITING TIME PLEASE:     1ØØ

TYPE NUMBER OF SAMPLE TIMES DESIRED:  5

T =  Ø.19732386E+Ø2
T =  Ø.684Ø147ØE+Ø2
T =  Ø.165Ø31Ø9E+Ø3
T =  Ø.36447561E+Ø1
T =  Ø.97657Ø15E+Ø2
TYPE MEAN WAITING TIME PLEASE:     5ØE+1

TYPE NUMBER OF SAMPLE TIMES DESIRED:  5

T =  Ø.54349454E+Ø3
T =  Ø.45728928E+Ø3
T =  Ø.579Ø14Ø5E+Ø3
T =  Ø.2274Ø226E+Ø3
T =  Ø.14563251E+Ø3



TYPE MEAN WAITING TIME PLEASE:     ↑C


.TECO RANDOM.MAC
*BJN$ACZ,2ØØØØ$IØ$ØLT$$
                TLO ACZ,2ØØØØØ  ;IN THE RANGE [Ø,1].
*EXIT$$
EXIT
↑C



.R PIP
*DT:RANDOM.MAC←DSK:RANDOM.MAC

*↑C



.DIRECT DT:
```

SAVE the core image of the two programs and DDT on DECtape. Assign this image file the name PROGA (an extension .SAV is automatically appended by the system).

As a double check, RUN the program you just saved on DECtape. Note that R is used to call in a program from the system device (SYS:) and RUN followed by a device name is used to call in a program from other devices.

Again, the results appear to be correct.

Use TECO to correct the Macro-10 source file to reflect the DDT correction.

Run Peripheral Interchange Program (PIP) to transfer the corrected Macro-10 source file to DECtape. At the end of every console session, it is suggested that the user transfer any files he might want to reuse from the disk area to tape or other storage medium. This procedure releases the disk space for other users and ensures that a refreshing of the disk will not destroy the only copy of a file.

Obtain a DIRECTory listing of the DECtape to ensure that it contains all the files you want to preserve. (DIRECT can be abbreviated as DIR.)

```
524. FREE BLOCKS LEFT
PROGA .SAV       27-JAN-69
RANDOM.MAC       27-JAN-69
ARRIVE.F4        22-JAN-69


EXIT
↑C


.KJOB
JOB 3, ONE OF USER 27,20 OFF TTY13 AT 1108 ON 27-JAN-69
FILES DELETED: 0, FILES SAVED: ALL , RUNTIME 0 MIN, 20 SEC

.
.AT 6 [27,20]



.K
CONFIRM: K
JOB 6, USER 27,20 OFF TTY13 AT 1118 ON 27-JAN-69
FILES DELETED: 11, FILES SAVED: 0, RUNTIME 0 MIN, 02 SEC
```

Kill the job. Monitor responds by printing the job number, project-programmer number; Teletype name; time, date, number of disk files deleted/saved; and the total run time.

ATTACH (or AT) to the second job (job #6), and kill it also. Following the CONFIRM: message, several options are available to determine what is to be done with the disk files. In this particular case, because all files that are to be preserved have already been stored on DECtape, type K to cause all disk files to be deleted.

Monitor prints the same information as in Step 13. You are now off the system; the core memory, disk space, and DECtape unit have been returned to the Monitor pool for others to use.

END OF DEMONSTRATION SESSION