digital

decsystem10 users handbook

second edition

Additional copies of this handbook may be ordered from:
Program Library, DEC, Maynard, Mass. 01754. Order code: DEC-10-NGZB-D.

decsystem10 handbook series

First Printing November 1971
Second Printing (Rev.) July 1972

tab index

NOTICE

For the reader's convenience:

1) Consecutive page numbers have been added to the top center of each page in the handbook; these numbers have the form –nn..– (for example –25–) and are supplied in addition to the standard document numbers printed at the bottom center of each page.

2) The appropriate document name has been added to the top outside corner of each page of the handbook.

3) A global index comprised of the merged and alphabetized entries of all of the indexes which were previously part of the documents contained by the handbook is supplied at the end of the handbook. The global index replaces the individual document indexes.

4) The entries of the global index and the Table of Contents for each document reference the *consecutive* page numbers located at the top center of each page.

5) Black locator tabs are printed on the outside edge of the first ten pages of each document in the handbook. A tab locator page on which each set of tabs is identified by the name of the document which they represent is supplied at the front of the handbook.

FOREWORD

This handbook is an introduction to the DECsystem-10. It is intended to be a guide to using the system and, as such, should be read before advancing to more detailed documentation. The collection of documents in this handbook is taken from the DECsystem-10 SOFTWARE NOTEBOOKS (DEC-10-SYZB-D) and in all cases, the documents are reprinted without change.

The documents in this handbook reflect the following versions of the software:

Monitor 5.05
TECO version 23
LINED version 13A
PIP version 32

Support program version numbers are specified on page 431 of this handbook.

The DECsystem-10 User's Handbook is one in the set of DECsystem-10 handbooks. The other handbooks comprising this series are:

(1) The COBOL Language Handbook and its supplement,

(2) The Mathematical Languages Handbook, which includes FORTRAN, BASIC, and ALGOL,

(3) The Assembly Language Handbook, which includes the System Reference Manual, MACRO, DECsystem-10 Monitor Calls, LOADER, DDT, CREF, FILCOM, FUDGE2, and GLOB.

These handbooks may also be ordered from the Program Library, Digital Equipment Corporation.

introduction

DEC-10-MZDA-D

decsystem10

INTRODUCTION TO DECsystem-10 SOFTWARE

DIGITAL EQUIPMENT CORPORATION · [illegible]

1st Printing November 1971

CONTENTS

CONTENTS (Cont)

CONTENTS (Cont)

ILLUSTRATIONS

FOREWORD

This manual is a general overview of the DECsystem-10. It is written for a person, not necessarily a full-time programmer, who is familiar with computers and computing systems and who desires to know the relationship between the various components of the computing system. Chapter 1 introduces the parts of DECsystem-10 along with the concept of multimode computing. Chapter 2 discusses the languages and utilities, those programs that enable the user to fully utilize the resources of the computing system. The more detailed Chapter 3 teaches the user the essential aspects of the resident operating system. Chapter 4 is an extensive glossary of terms used in describing both general and detailed ideas associated with the DECsystem-10. Appendix A is a survey of the DECsystem-10 hardware.

Introduction to DECsystem-10 Software is not intended to be a programmer's reference manual. For complete details on the operating system, the reader is referred to *TOPS-10 Operating System Commands*(DEC-10-MRDA-D) and *TOPS-10 Monitor Calls*(DEC-10-MRRA-D). However, it is recommended that this manual be read at least once before reading the foregoing manuals.

CHAPTER 1
THE DECsystem-10

The DECsystem-10 is more than a processor and its associated peripheral devices. Because it is a system, there are many parts, or components, working together to achieve a goal in a manner that is both convenient for the user of the system and advantageous for the operation of the system. It is a machine designed to be utilized concurrently by many users who wish to perform various operations. There are three major components of the computing system, as shown in Figure 1-1: the actual machine, or hardware; the operating system, or monitor; and the languages and utilities, or non-resident software.

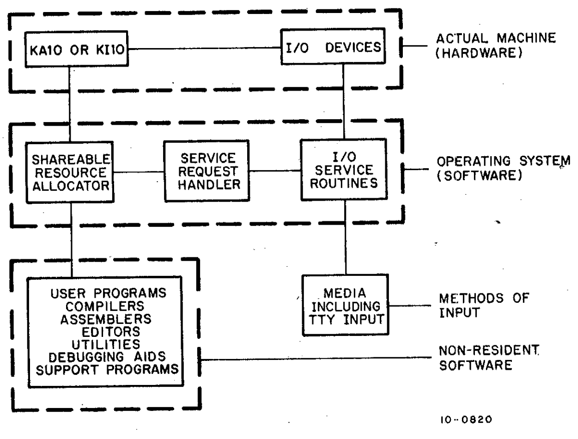
KA10 OR KI10
I/O DEVICES
ACTUAL MACHINE (HARDWARE)
SHAREABLE RESOURCE ALLOCATOR
SERVICE REQUEST HANDLER
I/O SERVICE ROUTINES
OPERATING SYSTEM (SOFTWARE)
USER PROGRAMS COMPILERS ASSEMBLERS EDITORS UTILITIES DEBUGGING AIDS SUPPORT PROGRAMS
MEDIA INCLUDING TTY INPUT
METHODS OF INPUT
NON-RESIDENT SOFTWARE
10-0820

Figure 1-1 DECsystem-10 Components

1.1 DECsystem-10 HARDWARE

The DECsystem-10 hardware consists of one or two central processors and various memories and input/output devices connected to these processors. There are five different systems included in the DECsystem-10 family, each system being distinguished by the hardware associated with the central processor. By adding hardware to an individual system, additional performance is achieved. However when adding hardware to expand from a small system to a larger system, no software changes are required. A single operating system and command control language can be used for all configurations of the DECsystem-10.

1.2 DECsystem-10 OPERATING SYSTEM

The DECsystem-10 hardware has numerous capabilities: it is powerful, fast, and highly sophisticated. Because of its complexity, this machine is not usually directly manipulated by its users. The users communicate with an intermediary, *the operating system*, in order to direct their problems to the actual machine and to receive solutions back. With many users on the system, this second component of the DECsystem-10 must also keep track of what each user does and the devices and system resources that each user accesses. Though the operating system cannot be seen like the actual machine, the action of the operating system is the most important and noticeable part of the system to each user. It is true that the operating system can do nothing for the user if the actual machine does not exist, but the user normally does not think of this. If the operating system accomplishes for him what he wants the actual machine to do,he is satisfied. Therefore, it is important to the user that he can depend on the same operating system regardless of the hardware that composes the actual machine.

The operating system is always resident in the core memory of the actual machine and is composed of three parts (refer to Figure 1-1). Because there are so many services that can be obtained from the operating system, including the allocation of core memory, processor time, and devices of the actual machine, one part, the *service request handler*, is responsible for accepting requests for these services. The service request handler passes the requests to another part, the *sharable resource allocator,* which is responsible for allocating the services requested. If the requested service is for use of a device, the *I/O service routines* are then notified to carry out the user's request.

1.3 DECsystem-10 NON-RESIDENT SOFTWARE

The third component of the DECsystem-10 is the non-resident software, those programs necessary for the varied operation of the computing system. This software includes the compilers, assemblers, editors, debugging programs, and operating system support programs. These software programs reside on a high-speed mass storage device of the actual machine and are brought into memory when needed by a user. By utilizing the non-resident software, the user of the computing system can create programs, transfer them from one device to another, compile, edit, execute, and debug them, and then receive the results of execution on any specified device.

1.4 DECsystem-10 MULTIPROCESSING

The DECsystem-10 can be a single-processor system or a dual-processor system, composed of a *primary processor* and a *secondary processor*. Each processor in the dual-processor system runs user programs, schedules itself, and fields instruction traps. In addition to these tasks, the primary processor also has control of all the input/output devices and processes all requests to the operating system. The primary processor completes any job that the secondary processor could not finish because of a request to the operating system. The two processors are connected to the same memory and execute the same copy of the operating system, thereby saving core memory over a multiprocessing system in which each processor has its own copy. The primary objective in the DECsystem-10 dual-processor environment is to provide more processing power than that found in the single-processor DECsystem-10. This means that with the addition of the second processor, more users can run at the same time. Or, if more users are not allowed on the system, the addition of the second processor reduces the elapsed time required to complete the processing of most programs.

1.5 MULTIMODE COMPUTING

The DECsystem-10 is designed for the concurrent operations of timesharing, multiprogram batch, real-time, and remote communications in either single or dual-processor systems. In providing these multifunction capabilities, the DECsystem-10 services interactive users, operates local and remote batch stations, and performs data acquisition and control functions for on-line laboratories and other real-time projects. By dynamically adjusting system operation, the DECsystem-10 provides many features for each class of user and is therefore able to meet a large variety of computational requirements.

1.5.1 Timesharing

Timesharing takes maximum advantage of the capabilities of the computing system by allowing many independent users to share the facilities of the DECsystem-10 simultaneously. Because of the interactive, conversational, rapid-response nature of timesharing, a wide range of tasks — from solving simple mathematical problems to implementing complete and complex information gathering and processing networks — can be performed by the user. The number of users on the system at any one time depends on the system configuration and the total computing load on the system. DECsystem-10 timesharing is designed to allow for up to 512 active terminals. These terminals include CRTs and other terminals which operate at speeds of 110 to 2400 baud. Terminal users can be located at the computer center or at remote locations connected to the computer center by communication lines.

1.5.1.1 Command Control Language — By allowing resources to be shared among users, the timesharing environment utilizes processor time and system resources that are wasted in single-user systems. Users are not restricted to a small set of system resources, but instead are provided with the full variety of facilities. By means of his terminal, the user has on-line access to most of the system's features. This on-line access is available through the operating system command control language, which is the means by which the timesharing user communicates with the computing system.

Through the command language, the user controls the running of his task, or job, to achieve the results he desires. He can create, edit, and delete his files; start, suspend, and terminate his job; compile, execute, and debug his program. In addition, since multiprogramming batch software accepts the same command language as the timesharing software, any user can enter his program into the batch run queue. Thus, any timesharing terminal can act as a *remote job entry terminal.*

1.5.1.2 Peripheral Devices — With the command language, the user can also request assignment of any peripheral device, e.g., magnetic tape, DECtape, and private disk pack, for his own exclusive use. When the request for assignment is received, the operating system verifies that the device is available to this user, and the user is granted its private use until he relinquishes it. In this way, the user can also have complete control of devices such as card readers and punches, paper-tape readers and punches, and line printers.

1.5.1.3 Spooling — When private assignment of a slow-speed device (e.g., card punch, line printer, paper-tape punch, and plotter) is not required, the user can employ the spooling programs of the operating system. *Spooling* is a method by which output to a slow-speed device is placed on a high-speed disk or drum. This technique prevents the user from using unnecessary time and space in core while waiting for either a device to become available or output to be completed. In addition, the device is managed to a better degree because the users cannot tie it up indefinitely, and the demand fluctuations experienced by these devices are equalized.

1.5.1.4 Mass Storage File System — Mass storage devices, such as disks and drums, cannot be requested for a user's exclusive use, but must be shared among all users. Because many users share these devices, the operating system must ensure independence among the users; one user's actions must not affect the activities of another unless the users desire to work together. To guarantee such independence, the operating system provides a *file system* for disks, disk packs, and drums. Each user's data is organized into groups of 128-word blocks called *files*. The user gives a name to each of his files, and the list of these names is kept by the operating system for each user. The operating system is then responsible for protecting each user's file storage from intrusion by unauthorized users.

In addition to allowing independent file storage for users, the operating system permits sharing of files among individual users. For example, programmers working on the same project can share the same data in order to complete a project without duplication of effort. The operating system lets the user specify *protection rights*, or *codes*, for his files. These codes designate if other users may read the file, and after access, if the files can be modified in any way.

The user of the DECsystem-10 is not required to preallocate file storage; the operating system allocates and deallocates the file storage space dynamically on demand. Not only is this convenient for the user because he does not have to worry about allocation when he is creating files, but this feature also conserves storage by preventing large portions of storage from being unnecessarily tied up.

1.5.1.5 Core Utilization — The DECsystem-10 is a *multiprogramming* system; i.e., it allows multiple independent user programs to reside simultaneously in core and to run concurrently. This technique of sharing core and processor time enhances the efficient operation of the system by switching the processor from a program that is temporarily stopped because of I/O transmission to a program that is executable. When core and the processor are shared in this manner, each user's program has a memory area distinct from the area of other users. Any attempt to read or change information outside of the area a user can access immediately stops the program and notifies the operating system.

Because available core can contain only a finite number of programs at any one time, the computing system employs a secondary memory, usually disk or drum, to increase the number of users serviced. User programs exist on the secondary memory and move into core for execution. Programs in core exchange places with the programs being transferred from secondary memory for maximum use of available core. Because the transferring, or *swapping*, takes place directly between core and the secondary memory, the central processor can be operating on a user program in one part of core while swapping is taking place in another. This independent overlapped operation greatly improves system utilization by increasing the number of users that can be accommodated at the same time.

To further increase the utilization of core, the operating system allows the users to share the same copy of a program or data segment. This prevents the excessive core usage that results when a program is duplicated for several users. A program that can be shared is called a *reentrant program* and is divided into two parts or *segments*. One segment contains the code that is not modified during execution (e.g., compilers and assemblers) and can be used by any number of users. The other segment contains the user's code and data that are developed during the compiling process. The operating system invokes protection for shared segments to guarantee that they are not accidentally modified.

1.5.1.6 General-Purpose Timesharing — Timesharing on the DECsystem-10 is general purpose; i.e., the system is designed in such a way that the command language, input/output processing, file structures, and job scheduling are independent of the programming language being used. In addition, standard software interfaces make it easy for the user to develop his own special languages or systems. This general purpose approach is demonstrated by the many programming languages implemented by DECsystem-10 customers.

1.5.2 Multiprogram Batch

Multiprogram batch software enables the DECsystem-10 to execute up to 14 batch jobs concurrently with timesharing jobs. Just as the timesharing user communicates with the system by way of his terminal, the batch user normally communicates by way of the card reader. (However, he can enter his job from an interactive terminal.) Unlike the timesharing user, the batch user can punch his job on cards, insert a few appropriate control cards, and leave his job for an operator to run. In addition, the user can debug his program in the timesharing environment and then run it in batch mode without any additional coding.

1.5.2.1 Multiprogram Batch Components — The multiprogram batch software consists of a series of programs: the Stacker, CDRSTK; the batch controller, BATCON; the queue manager, QMANGR; and the output spoolers, LPTSPL, CDPSPL, PTPSPL, and PLTSPL (see Figure 1-2). The *stacker* is responsible for reading the input from the input device and for entering the job into the batch controller's input queue. Although

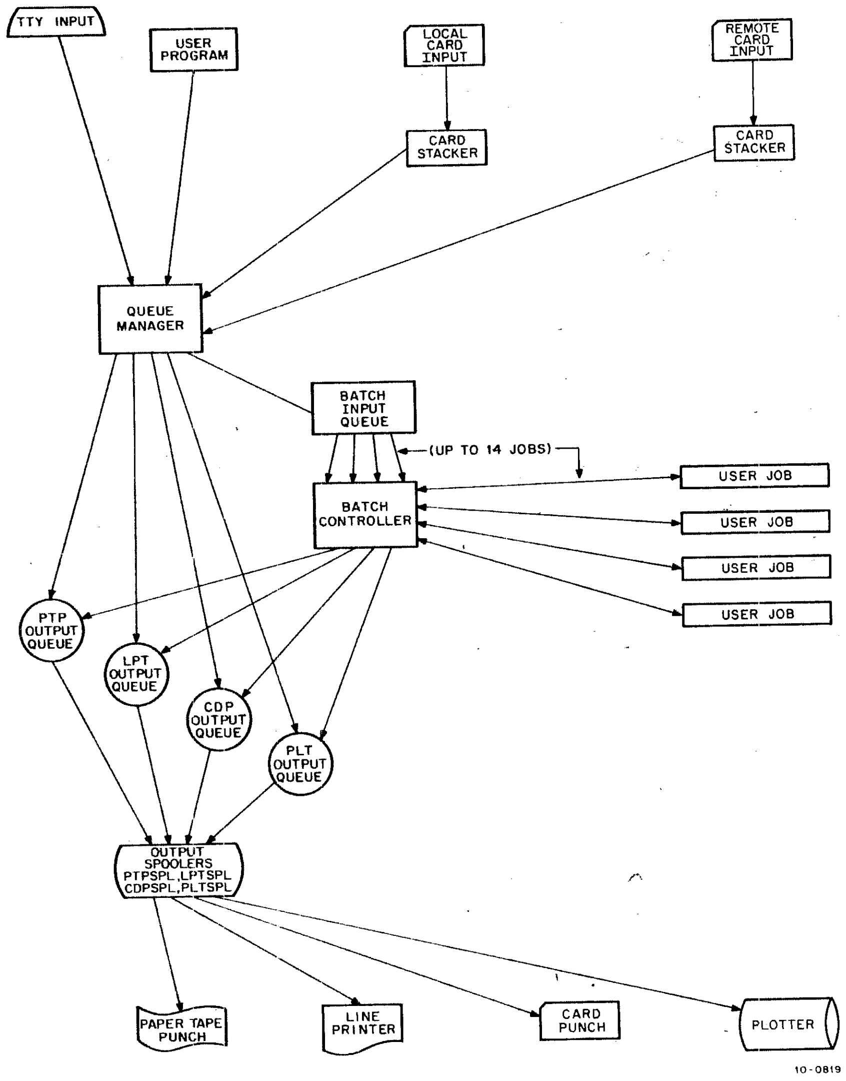

Figure 1-2 Programs in the Batch System

the Stacker is oriented toward card reader input, it allows jobs to be entered from any input device that supports ASCII code. The input information is then separated according to the control commands and placed into disk files, either user data files or the batch controller's control file, for subsequent processing. In addition, the Stacker creates the job's *log file* and enters a report of its processing of the job, along with a record of any operator intervention during its processing. The log file is part of the standard output that the user receives when his job terminates.

After the Stacker reads the end-of-file and closes the disk files, it makes an entry in the batch controller's input queue. The batch controller processes batch jobs by reading the entries in its queue. The control file created by the Stacker is read by the batch controller, and data and non-resident software commands are passed directly to the user's job. Operating system commands are detected by the batch controller and passed to the operating system for action. Most operating-system and non-resisdent-software commands available to the timesharing user are also available to the batch user. Therefore, only one control language need be learned for both timesharing and batch. During the processing of the job and the control file, the batch controller adds information to the log file for later analysis by the user.

The *queue manager* is responsible for scheduling jobs and maintaining both the batch controller's input queue and the output spooling queues. A job is scheduled to run under the batch controller according to external priorities, processing time limits, and core requirements which are dynamically computed by the queue manager, and according to parameters specified by the user for his job, such as start and deadline time limits for program execution. The queue manager makes an entry for the job in the batch input queue based upon the various priorities. After the job is completed, the queue manager again schedules it for output by placing an entry in an output queue. When the output is finished, the job's entry in the output queue is deleted by the queue manager.

The *output spooling programs* improve system throughput by allowing the output from a job to be written temporarily on the disk for later transfer instead of being written immediately on a particular output device. The log file and all job output are placed by the queue manager into one or more output queues to await spooling. When the specified device is available, the output is then processed by the appropriate spooling program. These spooling programs may be utilized by all users of the computing system.

1.5.2.2 Batch Use of System Features — The multiprogram batch software employs many of the computing system's features in order to operate with maximum efficiency. Because core memory is not partitioned between batch and timesharing jobs, batch jobs can occupy any available area of core. Fast throughput for high priority batch jobs is accomplished with the same swapping technique used for rapid response to interactive users. When available core is not large enough for a high priority batch job, the operating system transfers programs of lower priority to secondary memory in order to provide space for the job. This I/O transfer is done at the same time that the processor is operating on another job. Thus, processing can be overlapped with I/O to utilize time that would otherwise be wasted. Batch jobs can also share programs with timesharing and other batch jobs. Only one copy of a sharable program need be in core to service any number of batch and timesharing jobs at the same time.

1.5.2.3 Flexibility of the Batch System — Multiprogram batch allows the user a wide range of flexibility. The Stacker normally reads from the card reader, but can read from magnetic tape, DECtape, or disk packs in order to create a control file on disk and to enter the job into the batch controller's input queue. However, a job can be entered from an interactive terminal, in which case the user bypasses the Stacker and creates a control file on disk for the batch controller. The control file contains the operating system commands and non-resident software commands necessary to run the job. The user then enters the job into the batch controller's input queue by way of an operating system command string. In this command string, the user can include switches to define the operation and set the priorities and limits on core memory and processor time.

1.5.2.4 Job Dependency — Although jobs are entered sequentially into the batch system, they are not necessarily run in the order that they are read because of priorities, either set by the user in a stacker control command or computed by the queue manager when determining the scheduling of jobs. Occasionally, the user may wish to submit jobs that must be executed in a particular order; in other words, the execution of one job is dependent on another. To ensure that jobs are executed in the proper order, the user must specify an initial *dependency count* in a control command of the dependent job. This dependency count is then part of the input queue entry. A control command in the job on which the dependent job depends decrements the count. When the count becomes zero, the dependent job is executed.

1.5.2.5 Error Recovery — The user can control system response to error conditions by including in his job commands to the batch controller to aid in error recovery.

These commands are copied into the control file by the Stacker. With error recovery commands, the user specifies the action to be taken when his program contains a fatal error, as for example, to skip to the next program or to transfer to a special user-written error handling routine. If an error occurs and the user did not include error recovery conditions in his job, the batch controller initiates a standard dump of the user's core area and terminates the job. This core dump provides the user with the means to debug his program.

Although the batch system allows a large number of parameters to be specified, it is capable of operating with very few user-specified values. If a parameter is missing, the batch system supplies a reasonable default value. These defaults can be modified by the individual installations.

1.5.2.6 Operator Intervention — Normal operating functions performed by the programs in the batch system require little or no operator intervention; however, the operator can exercise a great deal of control if necessary. He can specify the number of system resources to be dedicated to batch processing by limiting the number of programs and both the core and processor time for individual programs. He can stop a job at any point, requeue it, and then change its priorities. By examining the system queues, he can determine the status of all batch jobs. In addition, the programs in the batch system can communicate information to the operator and record a disk log of all messages printed at the operator's console. All operator intervention during the running of the stacker and the batch controller causes messages to be written in the user's log file, as well as in the operator's log file, for later analysis.

1.5.3 Real-Time

For a system to be satisfactory for real-time applications, two important requirements must be met. The more important requirement is *fast response time*. Because real-time devices cannot store their information until the computing system is ready to accept it, the system would be useless for real-time if the response requirements of a real-time project could not be satisfied. The operating system must allocate system resources dynamically in order to satisfy the response and computational requirements of real-time jobs without affecting the simultaneous operations of timesharing and batch jobs. As part of its normal operation, the DECsystem-10 operating system satisfies this response requirement by overlapping I/O operations with processing time and by reacting to a constantly changing system load quickly and efficiently.

The second requirement is *protection*. Each user of the computing system must be protected from other users, just as the system itself is protected from all user program errors. In addition, since real-time systems have special real-time devices associated with jobs, the computing system must be protected from hardware faults that could cause system breakdown. And, because protection is part of the function of the operating system, the real-time software employs this feature to protect users as well as itself against hardware and software failures. Therefore, inherent in the operating system is the capability of real-time, and it is by way of calls to the operating system that the user obtains real-time services. The services obtained by calls within the user's program include 1) locking a job in core, 2) connecting a real-time device to the priority interrupt system, 3) placing a job in a high-priority run queue, 4) initiating the execution of FORTRAN or machine language code on receipt of an interrupt, and 5) disconnecting a real-time device from the priority interrupt system.

1.5.3.1 Locking Jobs — Memory space is occupied by the resident operating system and by a mix of real-time and non-real-time jobs. The only fixed partition is between the resident operating system and the remainder of memory. Since a real-time job needs to be in memory so as not to lose information when its associated real-time device interrupts, the job can request that it be *locked* into core. This means that the job is not to be swapped to secondary memory and guarantees that the job is readily available when needed. The operating system optimizes the placement of the job by positioning it in core so as to obtain the maximum amount of contiguous core space in the remaining memory. Because memory is not divided into fixed partitions, it can be utilized to a better degree by dynamically allocating more space to real-time jobs when real-time demands are high. As real-time demands lessen, more memory can be made available to timesharing and batch usage.

1.5.3.2 Real-Time Devices — The real-time user can connect real-time devices to the priority interrupt system, respond to these devices at interrupt level, remove the devices from the interrupt system, and/or change the priority interrupt level on which these devices are assigned. There is no requirement that these devices be connected at system generation time. The user specifies both the names of the devices generating the interrupts and the priority levels on which the devices function. The operating system then links the devices to the interrupt system.

The user can control the real-time device in one of two ways: *single mode* or *block mode*. In single mode, the user's interrupt program is run every time the real-time device interrupts. In block mode, the user's interrupt program is run after an entire block of data has been read from the real-time device. When the interrupt occurs from the device in single mode or at the end of a block of data in block mode, the operating system saves the current state of the machine and jumps to the user's interrupt routine. The user services his device and then returns control to the operating system to restore the previous state of the machine and to dismiss the interrupt. Any number of real-time devices may be placed on any available priority interrupt channel.

1.5.3.3 High-priority Run Queues — The real-time user can receive faster response by placing jobs in high-priorty run queues. These queues are examined before all other run queues in the computing system, and any runnable job in a high-priority queue is executed before jobs in other queues. In addition, jobs in high-priority queues are not swapped to secondary memory until all other queues have been scanned. When jobs in a high-priority queue are to be swapped, the lowest priority job is swapped first and the highest priority job last. The highest priority job swapped to secondary memory is the first job to be brought into core for immediate execution. Therefore, in addition to being scanned before all other queues for job execution, the high-priority queues are examined after all other queues for swapping to secondary memory and before all other queues for swapping from secondary memory.

1.5.3.4 Job Communication — The DECsystem-10 operating system enables a real-time user to communicate with other jobs through the use of *sharable data areas*. This also enables a data analysis program, for example, to read or write an area in the real-time job's core space. Since the real-time job associated with the data acquisition would be locked in core, the data analysis program residing on secondary memory would become core resident only when the real-time job had filled a core buffer with data. Operating system calls can be used to allow the data analysis program to remain dormant on secondary memory until a specified event occurs in the real-time job, e.g., a buffer has been filled with data for the data analysis program to read. When the specified event occurs, the dormant program is then activated to process the data. The core space for the real-time job's buffer area or the space for the dormant job does not need to be reserved at system generation time. The hardware working in conjunction with the operating system's core management facilities provides optimum core usage.

1.5.4 Remote Communications

Until the capability of remote communications was implemented, each remote user of the PDP-10 had been individually linked to the computer center by separate long distance telephone lines. Also, the only device that the remote user had available at his location was the terminal; he was able to utilize available devices at the central station, but he could not obtain output other than his terminal output at his remote site. Either he had to travel to the central station to obtain a listing or he had to have the listings delivered to him. However, with remote communications hardware and software, listing files and data can be sent via a single synchronous full-duplex line to a small remote computer, which in turn services many remote peripherals, including terminals, card readers, and line printers. This eliminates the need for the user to travel to the central site to obtain his output. The remote computer and its associated peripherals constitute a *remote station*.

Remote station use of the central computer is essentially the same as local use. All sharable programs and peripherals available to local users at the central computer station are also available to remote users. The remote user specifies the resources he wants to use and, if available, they are then allocated in the same manner as to a local user. In addition to utilizing the peripherals at the central station, the remote user can access devices located at his station or at another remote station. Local users at the central station can also make use of the peripherals at remote stations. Therefore, by specifying a station number in appropriate commands to the operating system, each user of the DECsystem-10 is given considerable flexibility in allocating system facilities and in directing input and output to the station of his choice.

The DECsystem-10 allows for simultaneous operation of multiple remote stations. Software provisions are incorporated in the operating system to differentiate one remote station from another. By utilizing peripheral devices at various stations, the user is provided with increased capabilities. For example, data can be collected from various remote stations, compiled and processed at the central station, and then the results of the processing can be sent to all contributors of the data.

Operating system commands not only allow a user to access peripherals at other remote stations, but also allow him to pretend that his job is at a remote station different from the physical station at which he is actually located. In this case, the user has a *logical station* and can run entire jobs from this station. With this capability, a local user at the central station could become a remote user as far as the system was concerned by changing the location of his job to a remote station in contact with the central station.

In summary, any computer, regardless of how powerful, is only as good as the operating system that maximizes its capabilities. The DECsystem-10 enhances the speed, power, and flexibility of the PDP-10 by dynamically responding to the changing user load and, in doing so, provides the user with a truly flexible and easily-used computing system.

CHAPTER 2
NON-RESIDENT SYSTEM SOFTWARE

For the computer to execute any of the basic operations which it is capable of executing, it must be told which operation it is to perform and where to find the information on which to perform the operation. This requires that a language be established by which the user can indicate to the computer what it needs to know. This language is the *machine language* of the computer and is unique for each machine. This machine language is the means by which the computer's circuits are instructed to perform the desired operation and because of this, it is the fastest and most direct method of programming. However, machine language programming is not the easiest method of programming for most users to employ. Although it is not impossible to memorize all of the operation codes recognized by the computer, it can be very difficult for the programmer to remember where each piece of information is inside memory in order to direct the computer to it. Therefore, *symbolic language programming* was developed to aid the programmer in manipulating the computer.

With symbolic language programming, programs are written using symbols which, when translated, equal the machine language of the computer. Symbol operation codes (mnemonics that specify which operation the user wants the computer to perform) are translated to the actual, or absolute, operation codes that the computer understands. Addresses of core are designated with symbolic labels and are converted into absolute core addresses so that the computer can locate the information on which to perform the desired operation.

There are three kinds of translators used in symbolic language programming: assemblers, compilers, and interpreters. An *assembler* is a program that is able to take another program written in symbolic language and translate it, item by item, into machine language. Therefore, to assemble a program means to substitute one absolute value for one symbolic notation until the entire program has been translated. A *compiler* also translates a symbolic language program into a machine language program, but the substitution is not one-to-one. A program written in a compiler language is freer in format than an assembly language program, and the language elements usually resemble English words. The compiler is larger and more complex than most assemblers, because it translates a program that is farther away from the machine language. Generally, one statement written in a compiler language is translated into several machine language instructions. Although a compiler occupies more space in memory and is generally slower than an assembler, a program written in a compiler language is more compatible with other computer models, and the language itself is easier to learn and write because of its general statements and freer format. An *interpreter* differs from an assembler or a compiler in that a binary version of the program is not produced for storage. In other words, the source text is translated to machine language everytime it is used, allowing for extensive checking of errors during execution.

2.1 DECsystem-10 ASSEMBLER

MACRO is the symbolic assembly program on the DECsystem-10. It makes machine language programming easier and faster for the user by (1) translating symbolic operation codes in the source program into the binary codes needed in machine language instructions, (2) relating symbols specified by the user to numeric values, (3) assigning absolute core addresses to the symbolic addresses of program instructions and data, and (4) preparing an output listing of the program which includes any errors detected during the assembly process.

MACRO programs consist of a series of statements that are usually prepared on the user's terminal with a system editing program. The elements in each statement do not have to be placed in certain columns nor must they be separated in a rigid fashion. The assembler interprets and processes these statements, generates binary instructions or data words, and performs the assembly.

MACRO is a two-pass assembler. This means that the assembler reads the source program twice. Basically, on the first pass, all symbols are defined and placed in the symbol table with their numeric values, and on the second pass, the binary (machine) code is generated. Although not as fast as a one-pass assembler, MACRO is more efficient in that less core is used in generating the machine language code and the output to the user is not as long.

MACRO is a device-independent program; it allows the user to select at runtime standard peripheral devices for input and output files. For example, input of the source program can come from the user's terminal and output of the assembled binary program can go to a magnetic tape, and output of the program listing can go to the line printer.

The MACRO assembler contains powerful macro capabilities that allow the user to create new language elements. This capability is useful when a sequence of code is used several times with only the arguments changed. The code sequence is defined with dummy arguments as a macro instruction. Thus, a single statement in the source program referring to the macro by name, along with a list of the real arguments, generates the correct and entire sequence. This capability allows for the expansion and adaptation of the assembler in order to perform specialized functions for each programming job.

2.2 DECsystem-10 COMPILERS

2.2.1 ALGOL

The ALGOrithmic Language, ALGOL, is a scientific language designed for describing computational processes, or algorithms. It is a problem-solving language in which the problem is expressed as complete and precise statements of a procedure.

The DECsystem-10 ALGOL system is based on ALGOL-60. It is composed of the ALGOL processor, or compiler, and the ALGOL object time system. The compiler is responsible for reading programs written in the ALGOL language and converting these programs into machine language. Also any errors the user made in writing his program are detected by the compiler and passed on to the user.

The ALGOL object time system provides special services, including the input/output service, for the compiled ALGOL program. Part of the object time system is the ALGOL library -a set of routines that the user's program can call in order to perform calculations. These include the mathematical functions and the string and data transmission routines. These routines are loaded with the user's program when required; the user need only make a call to them. The remainder of the object time system is responsible for the running of the program and providing services for system resources, such as core allocation and management and assignment of peripheral devices.

2.2.2 BASIC

The Beginner's All-purpose Symbolic Instruction Code, BASIC, is a problem-solving language that is easy to learn because of its conversational nature. It is particularly suited to a timesharing environment because of the ease of interaction between the user and the computer. This language can be used to solve problems with varying degrees of complexity, and thus, has wide application in the educational, business, and scientific markets.

BASIC is one of the simplest of the programming compiler languages available because of the small number of clearly understandable and readily learned commands that are required for solving almost any problem. The BASIC language is divided into two sections: one section of elementary commands that the user must know in order to write simple programs and the second section of advanced techniques for efficient and well-organized programs.

The BASIC user types in computational procedures as a series of numbered statements that are composed of common English terms and standard mathematical notation. When the statements are entered, a run-type command initiates the execution of the program and returns the results almost instantaneously.

The BASIC system has several special features built into its design. For one, BASIC contains its own editing facilities. Existing programs and data files can be modified directly with BASIC instead of with a system editor by adding or deleting lines, by renaming the file, or by resequencing the line numbers. The user can combine two files into one and request a listing of all or part of the file on either the line printer or the terminal. Secondly, BASIC allows various peripheral devices to be used for storage or retrieval of programs and data files. The user can input programs or data files from the paper-tape reader on the terminal or output them to the terminal's paper-tape punch. Also, the data file capability allows a program to read information from or write information to the disk. Thirdly, output to the terminal can be formatted by including tabs, spaces, and columnar headings. Finally, BASIC has an expanded command set that includes commands designed exclusively for matrix computations. Elementary mathematical functions are contained in the command set along with methods by which the user can define his own functions.

2.2.3 COBOL

The COmmon Business Oriented Language, COBOL, is an industry-wide data processing language that is designed for business applications, such as payroll, inventory control, and accounts-receivable.

Because COBOL programs are written in terms that are familiar to the business user, he can easily describe the formats of his data and the actions to be performed on this data in simple English-like statements. Therefore, programming training is minimal, COBOL programs are self-documenting, and programming of desired applications is accomplished quickly and easily.

The COBOL system is composed of a number of software components. The first is the COBOL compiler which is responsible for initializing the program, scanning the command strings for correct syntax, generating the code, listing, and final assembly. The second component is the object time system, LIBOL, which consists of subroutines used by the code generated by the compiler. These subroutines include the I/O, conversion, comparison, and mathematical routines available to the COBOL user. Another component is the source library maintenance program, which builds and maintains a library of source language entries that can be included in the user's source program at compile time. A fourth component is the stand-alone report generator, COBRG, which produces COBOL source programs, which when compiled and loaded, generate reports. The stand-alone program, SORT, accepts commands from the user's terminal in order to perform simple sorting of files. The RERUN program is used to restart a COBOL program that was interrupted during execution because of a system failure, device error, or disk quota error. COBDDT is a utility that debugs COBOL programs. Finally, ISAM builds and maintains indexed sequential files for the user.

DECsystem-10 COBOL accepts two source program formats: conventional format and standard format. The *conventional format* is employed when the user desires his source programs to be compatible with other COBOL compilers. This is the format normally used when input is from the card reader. The *standard format* is provided for users who are familiar with the format used in DECsystem-10 operations. It differs from conventional format in that sequence numbers and identification are not used because most DECsystem-10 programs require neither. The compiler assumes that the source program is written in standard format unless the appropriate switch is included in the command string to the compiler or the special sequence numbers created by the symbolic editor LINED are detected by the compiler.

DECsystem-10 COBOL is the highest level of ANSI COBOL available and because it operates within the operating system, it offers the user the many features of the DECsystem-10 in addition to the business processing capability of the language. These features enable the COBOL user to run programs in either, or both, timesharing or batch processing environments, to perform on-line editing and debugging of his programs with the system programs available, to choose various peripheral devices for input and output, and to write programs that can be shared with other users.

2.2.4 FORTRAN

The FORmula TRANSlator language, FORTRAN, is the most widely used procedure-oriented programming language. It is designed for solving scientific-type problems and thus is composed of mathematical-like statements constructed in accordance with precisely formulated rules. Therefore, programs written in the FORTRAN language consist of meaningful sequences of these statements that are intended to direct the computer to perform the specified computations.

FORTRAN has a varied use in every segment of the computer market. Universities find that FORTRAN is a good language with which to teach students how to solve problems via the computer. Scientific markets rely on FORTRAN because of the ease in which scientific problems can be expressed. In addition, FORTRAN is used as the primary data processing language by timesharing utilities.

Because of this wide market, DECsystem-10 FORTRAN is designed to meet the needs of all users. The FORTRAN system is easy to use in either the timesharing or batch processing environments. Under timesharing, the user operates in an interactive editing and debugging environment. Under batch processing, the user submits his program through the multiprogram batch software in order to have the compiling, loading, and executing phases performed without his intervention.

FORTRAN programs can be entered into the FORTRAN system from a number of devices: disk, magnetic tape, DECtape, user terminal, paper-tape reader, and card reader. In addition to data files created by FORTRAN, the user can submit data files or FORTRAN source files created by the system programs LINED, PIP, or TECO. The data files contain the data needed by the user's object program during execution. The source files contain the FORTRAN source text to be compiled by the FORTRAN compiler. Commands are entered directly to the FORTRAN compiler with a run-type command or indirectly through a system utility program that accepts and interprets the user's command string and passes it to the compiler. Output can then be received on the user's terminal, disk, DECtape, magnetic tape, card punch, or paper-tape punch.

2.3 DECsystem-10 INTERPRETER

The Algebraic Interpretive Dialogue, AID, is the DECsystem-10 adaptation of the language elements of JOSS, a program developed by the RAND Corporation. To write a program in the AID language requires no previous programming experience. Commands to AID are typed in via the user's terminal as imperative English sentences. Each command occupies one line and can be executed immediately or stored as part of a routine for later execution. The beginning of each command is a verb taken from the set of AID verbs. These verbs allow the user to read, store, and delete items in storage; halt the current processing and either resume or cancel execution; type information on his terminal; and define arithmetic formulas and functions for repetitive use that are not provided for in the language. However, many common algebraic and geometric functions are provided for the user's convenience.

The AID program is device-independent. The user can create external files for storage of subroutines and data for subsequent recall and use. These files may be stored on any retrievable storage media, but for accessibility and speed, most files are stored on disk.

2.4 DECsystem-10 EDITORS

2.4.1 LINED

The line editor for disk files, LINED, is used to create and edit source files written in ASCII code with line numbers appended. These line numbers allow LINED to reference a line in the file at any time without having the user close and then reopen the file. The user has the option of either specifying the beginning line number and the increment to the next line number when inserting lines or allowing LINED to assume a beginning line number and increment if the user specification is omitted.

Commands to LINED allow the user to create a new file or edit an existing file by inserting, replacing, or deleting lines within the file. Single or multiple lines of the file can be printed on the user's terminal for an aid in editing. When the user has the file as he desires, he closes the file and can either open a new file or return to monitor level to assemble or compile his file.

2.4.2 TECO

The Text Editor and COrrector program, TECO, is a powerful editor used to edit any ASCII text file with a minimum of effort. TECO commands can be separated into two groups: one group of elementary commands that can be applied to most editing tasks, and the larger set of sophisticated commands for character string searching, text block movement, conditional commands, programmed editing, and command repetition.

TECO is a character-oriented editor. This means that one or more characters in a line can be changed without retyping the remainder of the line. TECO has the capability to edit any source document: programs written in MACRO, FORTRAN, COBOL, ALGOL, or any other source language; specification; memoranda; and other types of arbitrarily-formatted text. The TECO program does not require that line numbers or other special formatting be associated with the text.

Editing is performed by TECO via an *editing buffer*, which is a section within TECO's core area. Editing is accomplished by reading text from any device (except a user's terminal) into the editing buffer (*inputting*), by modifying the text in the buffer with data received from either the user's terminal or a command file (*inserting*), and by writing the modified text in the buffer to an output file (*outputting*).

A position indicator, or *buffer pointer*, is used to locate characters within the buffer and its position determines the effect of many of TECO's commands. It is always positioned before the first character, between two characters, or after the last character in the buffer. Various commands, such as insertion commands, always take place at the current position of the buffer pointer.

Commands to TECO manipulate data within the editing buffer. Input and output commands read data from the input file into the buffer and output data from the buffer to the output file. One or more characters can be inserted into the editing buffer, deleted from the buffer, searched for, and or typed out with commands from the user at his terminal. In addition, the user can employ iteration commands to execute a sequence of commands repeatedly and conditional execution commands to create conditional branches and skips.

2.4.3 SOUP

The SOftware Updating Package, SOUP, is a set of programs that facilitates the updating of system or user source files. Because software is constantly being updated to reflect changes and improvements made by DEC, a method to make the updating process easier and faster for all concerned was developed. SOUP enables DEC to distribute a file containing only the differences to the software routine instead of redistributing the entire routine. In addition, since customers frequently maintain system files that are modified to reflect their individual needs, SOUP can be used to update these modified files as well. Although SOUP was implemented to update system files, it can be employed to update any source file with more than one version.

The SOftware Updating Package consists of three programs. The first program, CAM, is responsible for 1) comparing the new version of DEC's system file to the previous version to produce a correction file, and 2) merging two correction files derived from the same system file to produce a single correction file. The correction file contains a series of editing changes that reflect the differences between the old and new versions of the system files. The two functions of CAM can be performed simultaneously or one at a time depending on the user's command string to CAM.

The second program, COMP10, is used when the customer has modified DEC's file to such an extent that CAM cannot compare the modified file to the original file due to buffer overflow. COMP10 has extremely large buffers and can, therefore, be used to generate the correction file.

The third program, FED, reads the correction file and edits the copy of the system file by making the changes indicated in the correction file. When FED has completed its processing, the user has an updated file. As a software manufacturer, DEC sends the user a correction file, and he, in turn, need only run the FED program in order to update his system files.

2.4.4 RUNOFF

RUNOFF facilitates preparing typed or printed manuscripts by performing line justification, page numbering, titling, indexing, formatting, and case shifting as directed by the user. The user creates a file with TECO or LINED and inputs his material through his terminal. In addition to inputting the text, the user includes information for formatting and case shifting. RUNOFF processes the file and produces the final formatted file to be output to the terminal, the line printer, or to another file.

With RUNOFF, large amounts of material can be inserted into or deleted from the file without retyping the unchanged text. After the group of modifications have been added to the file, RUNOFF produces a new copy of the file which is properly paged and formatted.

2.5 DECsystem-10 UTILITIES

2.5.1 CREF

The cross-reference listing program, CREF, is an aid in program debugging and modification. It produces a sequence-numbered assembly listing followed by tables showing cross-references of all operand-type symbols, all user-defined operators, and all machine op codes and pseudo-op codes.

The input to CREF is a modified assembly listing created during assembly or compilation. The command string entered by the user specifies the device on which this assembly listing is located along with the output device on which to list the cross-reference tables and assembly listing. Switches can also be included in the command string in order to control magnetic tape positioning and to select specific sections of the listing output.

2.5.2 DDT

The Dynamic Debugging Technique, DDT, is used for on-line program composition of object programs and for on-line checkout and testing of these programs. For example, the user can perform rapid checkout of a new program by making a change resulting from an error detected by DDT and then immediately executing that section of the program for testing.

After the source program has been compiled or assembled, the binary object program with its table of defined symbols is loaded with DDT. In command strings to

DDT, the user can specify locations in his program, or *breakpoints*, where DDT is to suspend execution in order to accept further commands. In this way, the user can ckeck out his program section-by-section and if an error occurs, insert the corrected code immediately. Either before DDT begins execution or at breakpoints, the user can examine and modify the contents of any location. Insertions and deletions can be in source language code or in various numeric and text modes. DDT also performs searches, gives conditional dumps, and calls user-coded debugging subroutines at breakpoint locations.

2.5.3 File Backup

The file backup system enables the user to recover from a system failure or other unintentional destruction of data on the disk by 1) preserving disk files on a storage medium and 2) later retrieving these files and placing them back onto the disk. Two system programs are involved in this storage and retrieval system: the BACKUP program used to save the disk files on the specified storage device, and the RESTORE program used to return these files to the disk. Using the BACKUP program, the user can save individual disk files or the entire disk on magnetic tape, DECtape, or disk. When restoring these saved files to the disk with the RESTORE program, the user can return the entire contents of the storage device to the disk or return only selected portions.

2.5.4 FILEX

The file transfer program, FILEX, converts between various core image formats and reads or writes various DECtape directory formats and standard disk files. Files are transferred as 36-bit binary data with no processing performed on the data except that necessary to convert the core image representation. The core image formats that can be used in conversions are: 1) saved-file format, 2) expanded core image file format, 3) dump format, 4) simple block format (Project MAC's equivalent of DEC's .SAV format), and 5) binary file format. The desired core image format is determined by the specific extension associated with the file but this extension may be overridden by the use of switches in command strings to FILEX.

DECtapes can be read or written in binary, PDP-6 DECtape format, MIT Project MAC PDP-6/10 DECtape format, PDP-11, or PDP-15 format. In the latter two cases, ASCII files will be converted. The DECtape can be processed quickly via a *disk scratch file*, which is a much faster method for a tape with many files. This scratch file can be preserved and reused in later command strings. In addition, the DECtape directory can be listed on the user's terminal or zeroed in the appropriate format on the tape. These DECtape format and processing specifiers are indicated by command string switches.

2.5.5 LOADER

The LOADER provides automatic loading and relocation of binary programs generated by the standard DEC compilers and assemblers, produces an optional storage map, and performs loading and library searching regardless of the input medium. In addition, this program loads and links relocatable binary programs generated by the compilers and assemblers prior to execution and generates a symbol table in core for execution with DDT.

The user specifies in the LOADER command string the device from which the relocatable binary programs are to be loaded and the device on which any storage maps or undefined globals are written. Switches can be included in the command string 1) to specify the types of symbols to be loaded or listed, 2) to indicate that the run time libraries are to be searched for symbol definitions, 3) to load the DDT program, and 4) to clear and restart the LOADER. In addition, special switches allow the user to create CHAIN files—a feature used to segement FORTRAN programs that are too large to be loaded into core as one unit. These CHAIN files consist of complete programs and subroutines that can be read into core and executed as needed.

When the loading process is completed, the loaded program can be written onto an output device with a monitor SAVE command so that it can be executed at a later time without rerunning the LOADER.

2.5.6 PIP

The Peripheral Interchange Program, PIP, is used to transfer data files from one I/O device to another. Commands to PIP are formatted to accept any number of input (source) devices and one output (destination) device. Files can be transferred from one or more source devices to the destination device as either one combined file or individual files. Switches contained in the command string to PIP provide the user with the following capabilities: 1) naming the files to be transferred, 2) editing data in any of the input files, 3) defining the mode of transfer, 4) manipulating the directory of a device if it has a directory, 5) controlling magnetic tape and card punch functions, and 6) recovering from errors during processing.

2.6 DECsystem-10 MONITOR SUPPORT PROGRAMS

2.6.1 MONGEN

The monitor generator, MONGEN, is a dialogue program that enables the system programmer to define the hardware configuration of his individual installation and the set of software options that he wishes to select for his system. This program is a prerequisite for creating a new monitor.

The system programmer defines his configuration in one of four dialogues by answering MONGEN's questions in conversational mode. MONGEN transmits one question at a time to the user's terminal, and the user answers appropriately depending on the content of each question. After all questions have been answered, MONGEN produces MACRO source files containing the user's answers. These source files are then assembled and loaded with the symbol definition file and the monitor data base to yield a monitor tailored to the individual installation.

2.6.2 OPSER

The operator service program, OPSER, facilitates multiple job control from a single terminal by allowing the operator or the user to initiate several jobs, called *subjobs*, from his terminal. The OPSER program acts as the supervisor of the various subjobs by allowing monitor-level and user-level commands to be passed to all of the subjobs or to individually selected subjobs. Output from the various subjobs can then be retrieved by OPSER.

The subjobs of OPSER run on pseudo-TTYs, a simulated terminal not defined by hardware. All initializations of the pseudo-TTYs are performed by OPSER; the operator need only supply a subjob name. By running system programs, which ordinarily require a dedicated terminal, as subjobs of OPSER, output from the various programs can be concentrated on one hardware terminal instead of many. In addition, OPSER is able to maintain an I/O link between the running jobs and the operator—a feature that is not available when programs run on their own dedicated terminals.

2.6.3 LOGIN

LOGIN is the system program used to gain access to the DECsystem-10. This program determines by appropriate dialogue with the user who he is, whether or not he is currently authorized to use the system, and if so, establishes the user's initial profile, informs him of any messages of the day, and reports any errors detected in his disk files.

2.6.4 KJOB-LOGOUT

The system programs KJOB and LOGOUT are used when leaving the DECsystem-10. Their many functions include saving the user's disk files in the state in which he desires them, enforcing logged-out quotas on all disk file structures, terminating the user's job, and returning the resources allocated to the user back to the system. These resources include the user's job number, his allocated I/O devices, and his allocated core.

CHAPTER 3
THE RESIDENT OPERATING SYSTEM

The resident operating system is made up of a number of separate and somewhat independent parts, or routines (see Figure 3-1). Some of these routines are cyclic in nature and are repeated at every system clock interrupt (*tick*) to ensure that every user of the computing system is receiving his requested services. These cyclic routines are:

1) the command processor, or decoder
2) the scheduler, and
3) the swapper.

The *command decoder* is responsible for interpreting commands typed by the user on his terminal and passing them to the appropriate system program or routine. The *scheduler* decides which user is to run in the interval between the clock interrupts, allocates sharable system resources, and saves and restores conditions needed to start a program interrupted by the clock. The *swapper* rotates user jobs between secondary memory (usually disk or drum) and core memory after deciding which jobs should be in core but are not. These routines constitute the part of the operating system that allows many jobs to be operating simultaneously.

The non-cyclic routines of the operating system are invoked only by user programs and are responsible for providing these programs with the services available through the operating system. These routines are:

1) the UUO handler,
2) the input output routines, and
3) the file handler.

The *UUO handler* is the means by which the user program communicates with the operating system in order to have a service performed. Communication is by way of *programmed operators* (also known as *UUOs*) contained in the user program which, when executed, go to the operating system for processing. The *input / output routines* are the routines responsible for directing data transfers between peripheral devices and user programs in core memory. These routines are invoked through the UUO handler, thus saving the user the detailed programming needed to control peripheral devices. The *file handler* adds permanent user storage to the computing system by allowing users to store named programs and data as files.

3.1 THE COMMAND DECODER

The command decoder is the communications link between the user at his terminal and the operating system. Because all requests for system resources are initiated via the command decoder, it is the most visible part of the system to each user. When the user types commands and / or requests on his terminal, the characters are stored in an input buffer in the operating system. The command decoder examines these characters in the buffer, checks them for correct syntax, and invokes the system program or user program as specified by the command.

On each clock interrupt, control is given to the command decoder to interpret and process one command in the input buffer. The command appearing in the input buffer is matched with the table of valid commands accepted by the operating system. A match occurs if the command typed in exactly matches a command stored in the system, or if the characters typed in match the beginning characters of only one command. When the match is successful, the legality information (or flags) associated with the command is checked to see if the command can be performed immediately. For instance, a command can be delayed if the job is swapped out to the disk and the command requires that the job be resident in core; the command is executed on a later clock interrupt when the job is back in core. If all conditions as specified by the legality flags are met, control is passed to the appropriate program.

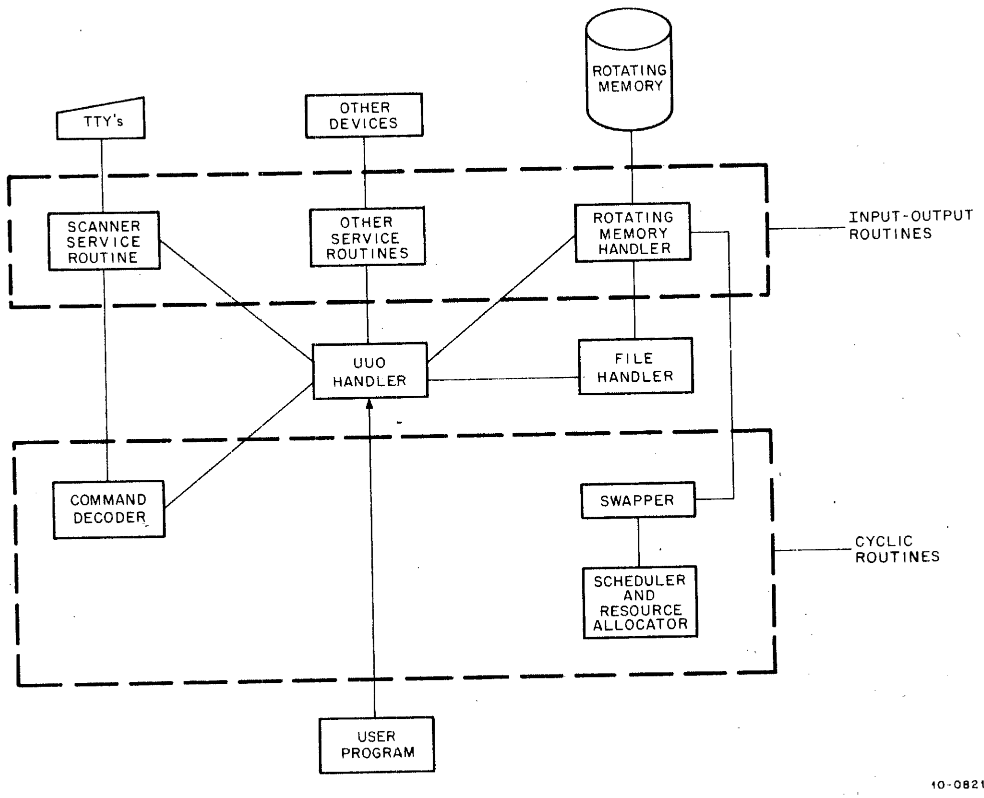

Figure 3-1 The Resident Operating System

3.2 THE SCHEDULER

The DECsystem-10 is a *multiprogramming* system; i.e., it allows several user jobs to reside in core simultaneously and to operate sequentially. It is then the job of the scheduler to decide which jobs should run at any given time. In addition to the multiprogramming feature, the DECsystem-10 employs a *swapping* technique whereby jobs can exist on an external storage device (e.g., disk or drum) as well as in core. Therefore, the scheduler decides not only what job is to be run next but also when a job is to be swapped out onto disk or drum and later brought back into core.

All jobs in the system are retained in ordered groupings called *queues*. These queues have various priorities that reflect the status of each job at any given moment. The queue in which a job is placed depends on the system resource for which it is waiting and, because a job can wait for only one resource at a time, it can be in only one queue at a time. Several of the possible queues in the system are:

1) run queues for jobs waiting for, or jobs in execution.
2) I/O wait queues for jobs waiting for data transfers to be completed.
3) I/O wait satisfied queues for jobs waiting to run after data transfers have been completed.
4) resource wait queues for jobs waiting for some system resource, and
5) null queue for all job numbers that are not currently being used.

The job's position within certain queues determines the priority of the job with respect to other jobs in the same queue. For example, if a job is first in the queue for a sharable device, it has the highest priority for the device when it becomes available. However, if a job is in an I/O wait queue, it remains in the queue until the I/O is completed. Therefore, in an I/O wait queue, the job's position has no significance. The status of a job is changed each time it is placed into a different queue.

The scheduling of jobs into different queues is governed by the *system clock*. This clock divides the time for the central processor into one-sixtieths of a second. Each job, when it is assigned to run, is given a *time slice* of a 1/2 second or two seconds, depending on the run queue. When the time slice expires for the job, the clock notifies the central processor and scheduling is performed. The job whose time slice just expired is moved into another run queue, and the scheduler selects the first job in the run queue to run in the next time slice.

Scheduling may be forced before the time slice has expired if the currently running job reaches a point at which it cannot immediately continue. Whenever an operating system routine discovers that it cannot complete a function requested by the job (e.g., it is waiting for I/O to complete or the job needs a device which it currently does not have), it calls the scheduler so that another job can be selected to run. The job that was stopped is then requeued and is scheduled to be run when the function it requested can be completed. For example: when the currently running job begins input from a DECtape, it is placed into the I/O wait queue, and the input is begun. A second job is scheduled to run while the input of the first job proceeds. If the second job then decides to access a DECtape, it is stopped because the DECtape control is busy, and it is placed in the queue for jobs waiting to access the DECtape control. A third job is set to run. The input operation of the first job finishes, freeing the DECtape control for the second job. The I/O operation of the second job is initiated, and the job is transferred from the device wait queue to the I/O wait queue. The first job is transferred from the I/O wait queue to the highest priority run queue. This permits the first job to preempt the running of the third job. When the time slice of the first job becomes zero, it is moved into the second run queue, and the third job runs again until the second job completes its I/O operations.

In addition, data transfers use the scheduler to permit the user to overlap computation with data transmission. In unbuffered data modes, the user supplies an address of a command list containing pointers to locations in his area to and from which data is to be transferred. When the transfer is initiated, the job is scheduled into an I/O wait queue where it remains until the device signals the scheduler that the entire transfer has been completed.

In buffered modes, each buffer contains information to prevent the user and the device from using the same buffer at the same time. If the user requires the buffer currently being used by the device as his next buffer, the user's job is scheduled into an I/O wait queue. When the device finishes using the buffer, the device calls the scheduler to reactivate the job.

3.3 THE SWAPPER

The swapper is responsible for keeping in core the jobs most likely to be runnable. It determines if a job should be in core by scanning the various queues in which a job may be. If the swapper decides that a job should be brought into core, it may have to take another job already in core and transfer it to secondary memory. Therefore, the swapper is not only responsible for bringing a job into core but is also responsible for selecting the job to be swapped out.

A job is swapped to secondary memory for one of two reasons: 1) a job that is more eligible to run needs to be swapped in and there is not enough room in core for both jobs, and 2) the job needs to expand its core size and there is not enough core space to do so. If the later case is true, the job must be swapped out and then swapped in later with the new allocation of core.

The swapper checks periodically to see if a job should be swapped in. If there is no such job, then it checks to see if a job is requesting more core. If there is no job wishing to expand its size, then the swapper does nothing further and waits until the next clock tick.

3.4 THE UUO HANDLER

The UUO handler is responsible for accepting requests for services available through the operating system. These requests are made by the user program via software-implemented instructions known as *programmed operators*, or *UUOs*. The various services obtainable by the user program include:

1) communicating with the I/O devices on the computing system, including connecting and responding to any special devices that may be desired on the system for real-time programming,
2) receiving or changing information concerning either the computing system as a whole or the individual program,

3) altering the operation of the computing system as it concerns the user job, such as controlling execution by trapping or suspending, or controlling core memory by locking,
4) communicating and transferring control between user programs.

The UUO handler is the only means by which a user program can give control to the operating system in order to have a service performed. Contained in the user program are operation codes which, when executed, cause the hardware to transfer control to the UUO handler for processing. This routine obtains its arguments from the user program. The core location at which the UUO operation was executed is then remembered. After the UUO request has been processed, control is returned to the user program at the first or second instruction following the UUO. In this way, the software supplements the hardware by providing services that are invoked through the execution of a single core location just as the hardware services are invoked.

3.5 THE INPUT/OUTPUT ROUTINES

I/O programming in the DECsystem-10 is highly convenient for the user because all of the burdensome details of programming are performed by the operating system. The user informs the operating system of his requirements for I/O by means of UUOs contained in his program. The actual input/output routines needed are then called by the UUO handler.

Since the operating system channels communication between the user program and the device, the user does not need to know all the peculiarities of each device on the system. In fact, the user program can be written in a similar manner for all devices. The operating system will ignore, without returning an error message, operations that are not pertinent to the device being used. Thus, a terminal file and a disk file can be processed identically by the user program. In addition, user programs can be written to be independent of any particular device. The operating system allows the user program to specify a *logical device name*, which can be associated with any physical device at the time when the program is to be executed. Because of this feature, a program that is coded to use a specific device does not need to be rewritten if the device is unavailable. The device can be designated as a logical device name and assigned to an available physical device with one command to the operating system.

Data is transmitted between the device and the user program in one of two methods: *unbuffered mode* or *buffered mode*. With unbuffered data modes, the user in his program supplies the device with an address, which is the beginning of a command list. Essentially, this command list contains pointers specifying areas in the user's allocated core to or from which data is to be transferred. The user program then waits until the operating system signals that the entire command list has been processed. Therefore, during this data transfer, the user program is idly waiting for the transfer to be completed.

Data transfers in buffered mode utilize a ring of buffers set up in the user's core area. Buffered transfers allow the user program and the operating system's I/O routines to operate asynchronously. As the user program uses one buffer, the operating system processes another one by filling or emptying it as interrupts occur from the device. To prevent the user program and the operating system from using the same buffer at the same time, each buffer has a *use bit* that designates who is using the buffer. Buffered data transfers are faster than unbuffered transfers because the user program and the operating system can be working together in processing the data.

Several steps must be followed by the user program in order for the operating system to have the information it needs to control the data transfers. Each step is indicated to the operating system with one programmed operator. In the first step, the specific device to be used in the data transfer must be selected and linked to the user program with one of the software I/O channels available to the user's job (OPEN or INIT programmed operators). This device remains associated with the software I/O channel until it is disassociated from it (via a programmed operator) or a second device is associated with the same channel. In addition to specifying the I/O channel and the device name, the user program can supply an initial file status, which includes the type of data transfer to be used with the device (e.g., ASCII, binary), and the location of the headers to be used in buffered data transfers. The operating system stores information in these headers when the user program executes programmed operators, and the user program obtains from these headers all the information needed to fill or empty buffers.

Another set of programmed operators (INBUF and OUTBUF) establishes the actual buffers to be used for input and output. This procedure is not necessary if the user is satisfied to accept the two buffers automatically set up for him by the operating system.

The next step is to select the file that the user program will be using when reading or writing data. This group of operators (LOOKUP and ENTER) is not required for devices that are not file-structured (e.g., card reader, magnetic tape, paper-tape punch); however, if used, they will be ignored thus allowing file-structured devices to be substituted for non-file-structured devices without the user rewriting the program.

The third step is to perform the data transmission between the user program and the file (IN, INPUT, OUT, and OUTPUT). When the data has been transmitted to either the user program on input or the file on output, the file must be closed (CLOSE, fourth step) and the device released from the channel (RELEASE, fifth step). This same sequence of programmed operators is performed for all devices; therefore, the I/O system is truly device independent because the user program does not have to be changed every time a different device is used.

In addition to reading or writing data to the standard I/O devices, provisions are included in the operating system for using the terminal for I/O during the execution of the user program. This capability is also obtained through programmed operators. As the user program is running, it can pause to accept input from or to type output to the terminal. The operating system does all buffering for the user, thus saving him programming time. This method of terminal I/O provides the user with a convenient way of interacting with his running program.

3.6 FILE HANDLER

The disk file handler manages user and system data; thus, this data can be stored, retrieved, protected, and/or shared amoung other users of the computing system. All information in the system is stored as *named files* in a uniform and consistent fashion thus allowing the information to be accessed by name instead of by physical disk addresses. Therefore, to reference a file, the user does not need to know where the file is physically located. A named file is uniquely identified in the system by a *filename* and *extension*, an ordered list of *directory names* (UFDs and SFDs) which identify the owner of the file, and a *file structure name* which identifies the group of disk units containing the file.

Usually a complete disk system is composed of many disk units of the same and or different types of disks. Therefore, the disk system consists of one or more file structures–a logical arrangement of files on one or more disk units of the same type. This method of file storage allows the user to designate which disk unit of the file structure he wishes to use when storing his files. Each file structure is logically complete and is the smallest section of file memory that can be removed from the system without disturbing other units in other file structures. All pointers to areas in a file structure are by way of logical block numbers rather than physical disk addresses; there are no pointers to areas in other file structures, thereby allowing the file structure to be removed.

A file structure consists of two types of files: the *data files* that physically contain the stored data or programs, and the *directory files* that contain pointers to the data files. Included in these directory files are master file directories, user file directories, and sub-file directories. Each file structure has one *master file directory* (MFD). This directory file is the master list of all the users of the file structure. The entries contained in the MFD are all the names of the user file directories on the file structure. Each user with access to the file structure has a *user file directory* (UFD) that contains the names of all his files on that file structure; therefore, there are many UFDs on each file structure. As an entry in the user file directory, the user can include another type of directory file, a *sub-file directory* (SFD). The sub-file directory is similar to the other types of directory files in that it contains as entries all the names of files within the directory. This third level of directory allows groups of files belonging to the same user to be separate from each other. This is useful when organizing a large number of files according to function. In addition, sub-file directories allow non-conflicting simultaneous batch runs of the same program using the same filenames.

As long as the files are in different sub-file directories, they are unique. Sub-file directories exist as files pointed to by the user file directory, and can be nested to the depth specified by the installation via a MONGEN question

All disk files are composed of two parts: data and information used to retrieve the data. The retrieval part of the file contains the pointers to the entire file, and is stored in two distinct locations on the device and accessed separately from the data. System reliability is increased with this method because the probability of destroying the retrieval information is reduced; system performance is improved because the number of positionings needed for random-access methods is reduced. The storing of retrieval information is the same for both sequential and random access files. Thus a file can be created sequentially and later read randomly, or vice versa, without any data conversion.

One section of the retrieval information is used to specify the *protection* associated with the file. This protection is necessary because disk storage is shared among all users, each of whom may desire to share files with, or prevent files from being written, read or deleted by, other users. These protection codes are assigned by the user when the file is created and designate the users who have privileges to access the file. Users are divided into three categories: the user who created the file (the owner of the file), the user on the same project as the owner of the file, and the remaining users of the system. The owner of the file controls the protection of the file; thus, he can indicate who may read, write, or modify his file. It is always possible for the owner to change the protection of his file and when it is changed, the new protection remains until he modifies it again. If a file is created without a protection code, the operating system substitutes an installation-defined standard protection code.

Disk *quotas* are associated with each user (each project-programmer number) on each file structure in order to limit the amount of information that can be stored in the UFD of a particular file structure. When the user gains access to the computing system, he automatically begins using his *logged-in quota*. This quota is not a guaranteed amount of space, and the user must compete with other users for it. When the user leaves the computing system, he must be within his *logged-out quota*. This quota is the amount of disk storage space that the user is allowed to maintain when he is not using the system and is enforced by the system program that is used in leaving the system. Quotas are determined by the individual installations and are, therefore, used to ration disk resources in a predetermined manner.

To a user, a file structure is like a device; i.e., a file structure name or a set of file structure names can be used as the device name in command strings or UUO calls to the operating system. Although file structures or the units composing the file structures can be specified by their actual names, most users specify a general, or generic, name (DSK) which will cause the operating system to select the appropriate file structure. The appropriate file structure is determined by a *job search list*. Each job has its own job search list with the file structure names in the order in which they are to be accessed when the generic name is specified as the device. This search list is established by LOGIN and thus each user has a UFD for his project-programmer number in each file structure in which LOGIN allows him to have files.

File writing on the disk can be defined by one of three methods: creating, superseding, and updating. The user is *creating* a file if no other file of the same name exists in the user's directory on the indicated file structure. If another file with the same name already exists in the directory, the user is *superseding*, or replacing, the old file with the new file. Other users sharing the old file at the time it is being superseded continue using the old file and are not affected until they finish using the file and then try to reaccess it later. At that time, they read the new file. When a user *updates* a file, he modifies selected parts of the file without creating an entirely new version. This method eliminates the need to recopy a file when making only a small number of changes. If other users try to access a file while it is being updated, they receive an error.

File storage is dynamically allocated by the file handler during program operations, so the user does not need to give initial estimates of file length or the number of files. Files can be any length, and each user may have as many files as he wishes, as long as disk space is available and the user has not exceeded his logged-in quota. This feature is extremely useful during program development or debugging when the final size of the file is still unknown. However, for efficient random access, a user can reserve a contiguous area on the disk if he desires. When he has completed processing, he can keep his pre-allocated file space for future use or return it so that other users can have access to it.

3.7 SUMMARY

In summary, the resident operating system supervises user jobs and provides various services to these jobs. It acts as an operator by performing specific functions in response to specific events which occur within the system. Many functions are performed in accordance with a periodic event, the system clock interrupt. Other functions are responded to in accordance with the action of the user program.

CHAPTER 4 GLOSSARY

Absolute address
The address that is permanently assigned to a storage location by the machine designer.

Access date
The date on which a file on disk was last read. If a file has not been read since it was created, the creation date and the access date are the same. The access date is kept in the retrieval information block for the file.

Access list
The table in monitor core that reflects the status of all files open for reading or writing in addition to the status of those files recently closed.

Access privileges
Attributes of a file which specify the class of users allowed to access the file and the type of access which they are allowed.

Access time
The interval between the instant at which data is requested from a storage device or data is requested for a storage device and the instant at which delivery or storage is begun.

ACCT. SYS
The file that contains all project-programmer numbers, passwords, initial profiles, and time of day users are allowed on the system. It does not contain file structure quotas.

Accumulator
The register and associated equipment in the arithmetic unit of the computer in which arithmetical and logical operations are performed.

Active search list
An ordered list of file structures for each job which specifies the order in which the directory is searched. These file structures are the ones listed before the FENCE by the SETSRC program. Device DSK is defined by this list for each job.

Actual transfer
The hardware operation whereby the channel actually passes data between the memory system and the control. The third step of the transfer operation (verification, search, actual transfer).

Address

(1) An identification represented by a name, label, or number for a register, a location in storage, or any other data source or destination.

(2) The part of an instruction that specifies the location of an operand of the instruction.

ALCFIL
A program used for allocating space for a new file or reallocating space for an existing file in one contiguous region on the disk.

ALGOTS
The ALGOL object time system.

All CPU job
A job which the monitor can run on either processor in a dual-processor system depending on the I/O activity and the system load.

Arithmetic unit
The portion of the hardware in which arithmetic and logical operations are performed.

Assemble
To prepare a machine-language program from a symbolic-language program by substituting absolute operation codes for symbolic operation codes and absolute or relocatable addresses for symbolic addresses.

Assembler
A program which accepts symbolic code and translates it into machine instruction, item by item.

Assigning a device
To allocate an I/O device to the user's job either for the duration of the job or until the user relinquishes it.

Asychronous

(1) Pertaining to the procedure by which the hardware does not wait for one operation to be completed before starting a second operation.

(2) Pertaining to the method of data transmission in which each character is sent with its own synchronizing information.

AUXACC.SYS
The file that contains the standard list of public file structures for each user and information (such as quotas) for those file structures.

Bad Allocation Table (BAT) block
A block written by the MAP program or the monitor on every disk unit. This block enumerates the bad regions of consecutive bad blocks on that unit so that they are not reused. The BAT blocks appear in the HOME.SYS file.

BADBLK.SYS
The file that contains all bad blocks. It may be read but not deleted and is useful for testing error recovery.

Base address
A given address from which an absolute address is derived by combination with a relative address.

Batch processing
The technique of executing a set of computer programs in an unattended mode.

BATCON
The Batch controller. This program reads a job's control file, starts the job, and controls the job by passing commands and data to it.

Block
A 128_{10}-word unit of disk storage determined by hardware and software; 128 words are always written, adding zeros as necessary, although less than 128 words can be read.

BOOTS
A bootstrap program whose main functions are to load a program into core from a disk SAVE file and to dump core as a SAVE file for later analysis.

Bootstrap
A technique or device designed to bring itself into a desired state by means of its own action, e.g., a machine routine whose first instructions are sufficient to bring the rest of itself into the computer from an input device.

Breakpoint
A location at which program operation is suspended in order to examine partial results.

Buffer
A device or area used temporarily to hold information being transmitted between external and internal storage devices or I/O devices and internal high-speed storage. A buffer is often a special register or a designated area of internal storage.

Buffer pointer
A movable position indicator that is positioned between two characters in an editing buffer, before the first character in the buffer, or after the last character in the buffer.

Byte
Any contiguous set of bits within a word.

Calling sequence
A specified arrangement of instructions and data necessary to pass parameters and control to and from a given subroutine.

CDRSTK
The Batch input stacker. CDRSTK reads any sequential input stream, sets up the job's control file and data files, and enters the job into the Batch input queue.

Central processing unit (CPU)
The portion of the computer that contains the arithmetic, logical, control circuits, and I/O interface of the basic system.

Central site
The location of the central computer. Used in conjunction with remote communications to mean the location of the DECsystem-10 central processor.

CHAIN
A program that allows the user to segment FORTRAN programs that are too large to load or fit into available core. It reads successive segments of coding into core and links them to the program already in core.

Channel
(1) A path along which signals can be sent; e.g., data channel, output channel.
(2) A partially autonomous portion of the PDP-10 which can overlap I/O transmission while computations proceed simultaneously.

CHECKPOINT
A program used to maintain the accounting information on the disk.

Clear
To erase the contents of a location by replacing the contents with blanks or zeros.

Cluster
A single-or multi-block unit of disk storage assignment. It is a parameter of each disk file structure.

CODE
A code conversion program that translates files written in binary-coded decimal to ASCII and vice versa.

COMPIL
A utility program that allows the user to type a short, concise command string in order to cause a series of operations to be performed. COMPIL deciphers the command and constructs new command strings for the system program that actually processes the command.

Compressed file pointer
An 18-bit pointer to the unit within the file structure and to the first super-cluster of the file.

Concatenation
The joining of two strings of characters to produce a longer string, often used to create symbols in macro defining.

Conditional jump
A jump that occurs if specified criteria are met.

Context switching
The saving of sufficient hardware and software information of a process so that it may be continued at a later time, and the restoring of another process.

Continued directory
The collection of all directories with a particular name and path on all file structures in the job's search list.

Continued MFD
The MFDs on all file structures in the job's search list.

Continued SFD
The SFDs on all file structures in the job's search list which have the same name and path.

Continued UFD
The UFDs for the same project-programmer number on all file structures in the job's search list.

Control
The device which controls the operation of connected units. It can initiate simultaneous positioning commands to some of its units and then perform a data transfer for one of its units.

Control character
A character with an ASCII representation of 0-37. It is typed by holding down the CTRL key on the terminal while striking a character key. It can be punched on a card via the multi-punch key.

Copy
To transfer a file from one device to another (e.g., with PIP or the FILEX program).

CORMAX
The largest contiguous size that an unlocked job can be. This value can range from CORMIN to total user core.

CORMIN
The quaranteed amount of contiguous core which a single unlocked job can have. This value can range from 0 to total user core.

Counter
A device such as a register or storage location used to represent the number of occurrences of a certain event.

CPU
See central processing unit.

CPU0
In a dual-processor system, the processor that performs the same activities as the processor in a single processor system, including all I/O operations, command and UUO processing, swapping, and interrupt handling. Also known as the primary processor.

CPU1
In a dual-processor system, the processor that operates only in user mode except when it is required to find another job to run or to send APR traps to the user. Also known as the secondary processor.

CRASH.SAV
A file written on disk by BOOTS as part of the crash restart procedure. This file is used by FILDDT for system debugging.

Create
To open, write, and close a file for the first time. Only one user at a time can create a file with a given name and extension in the same directory or sub-directory of a file structure.

CREF
A program which produces a sequence-numbered assembly listing followed by tables showing cross references for all operand-type symbols, all user-defined operators, and/or all op codes and pseudo-op codes.

Customer
A Digital customer purchasing a DECsystem-10 as distinguished from a user at a console who may be purchasing time from a customer.

Cylinder
The hardware-defined region of consecutive logical disk blocks which can be read or written without repositioning.

DAEMON
A program for writing all or parts of a job's core area and associated monitor tables onto disk.

Data Channel
The device which passes data between the memory system and the control.

DATDMP.
A program for dumping the core data base.

DECtape
A convenient, pocket-sized reel of random access magnetic tape developed by Digital Equipment Corporation.

DDT
The Dynamic Debugging Technique program used for on-line checkout, testing, and program composition of object programs.

Device routines
Routines that perform I/O for specific storage devices and translate logical block numbers to physical disk addresses. These routines also handle error recovery and ensure ease of programming through device independence.

DIRECT
A program for producing directory listings of disks and DECtapes.

Directory
A file which contains the names and the pointers to other files on the device. On disk, a directory is continued across all the file structures in a job's search list. Continued MFDs, UFDs, and SFDs are all directories. The DIRECT monitor command lists a directory.

Directory device
A storage retrieval device such as disk or DECtape which contains a file describing the layout of stored data (programs and other files).

Directory path
The ordered list of directory names, starting with a UFD name, which uniquely specifies a directory without regard to a file structure. Also known as a path. A file structure name, a path, and a filename and extension are needed to uniquely identify a file in the system.

Dismounting a file structure
The process of deleting a file structure from a user's active search list by using the DISMOUNT command. It does not necessarily imply physical removal of the file structure from the system.

Doorbell
The device by which processors in a multiprocessing system interrupt each other. This is an optional device.

Dormant file structure
A file structure that is physically mounted but has no current users, i.e., the mount count is zero.

Dormant segment
A sharable high segment kept on a swapping space, and possible core, which is in no user's addressing space.

DSK
The generic device name for disk-like devices. Actual file structure names are defined for each job by the file structure search list.

DSKLST
A program which gives statuses and statistics of all user disk files at a given point in time.

DSKRAT
A damage assessment pogram that scans a file structure and reports any inconsistencies detected.

Dump
A listing of all variables and their values, or a listing of the values of all locations in core.

DUMP
A program that outputs selected portions of a file in one of the various formats that can be specified by the user.

EDDT
A version of DDT used for debugging programs, such as the monitor, in executive mode.

Effective address
The actual address used, that is, the specified address as modified by any indexing or indirect addressing rules.

Entry point
A point in a subroutine to which control is transferred when the subroutine is called.

Executive mode
A central processor mode characterized by the lack of memory protection and relocation and by the normal execution of all defined operation codes.

Extended file
A file which contains one or more extended RIBs to contain the retrieval pointers.

Extended RIB
Additional retrieval information blocks (RIBs) required when the retrieval pointers in a file overflow the prime RIB.

FAILSAFE
A utility program used to save the contents of the disk on magnetic tape and later restore the saved contents back onto disk.

FILDDT
A version of DDT used for examining and changing a file on disk instead of in core memory. This program is used to examine a monitor for debugging purposes.

File
An ordered collection of 36-bit words comprising computer instructions and/or data. A file can be of any length, limited only by the available space on the device and the user's maximum space allotment on that device. A file is uniquely identified in the system by a file structure name or directory name, a directory path, and a filename and extension.

Filename
A name of one to six alphanumeric characters chosen by the user to identify a file.

Filename extension
One to three alphanumeric characters usually chosen by the program to describe the class of information in a file.

File specification
A list of quantities which uniquely identify a named file. A complete file specification consists of: the name of the physical device or file structure on which the file is stored, the name of the file including its extension, the name of the directory in which the file is contained, and the protection code associated with the file. File specifications are ignored for non-file-oriented devices.

File structure
The logical arrangement of 128-word blocks on one or more units of the same type to form a collection of named files.

File-structured device
A device on which data is given names and arranged into files; the device also contains directories of these names.

File structure owner
The user whose project-programmer number is associated with the file structure in the administrative file STRLST.SYS. The REACT program is used to enter or delete this project-programmer number or any of the other information that is contained in an STRLST.SYS. entry.

File structure search list
For each job, a list that specifies the order in which the file structures that user can access are to be searched when device DSK: is specified. Also called a job search list.

FILEX
A general file transfer program used to convert between various core image formats and to read and write various DECtape directory formats and standard disk files.

Flag
An indicator that signals the occurrence of some condition, such as the end of a word.

Fragmentation
The technique used when swapped segments cannot be allocated in one contiguous set of blocks on the swapping space.

FUDGE2
A file update generator used to update files containing one or more relocatable binary programs and to manipulate programs within program files.

Full path name
The ordered list which uniquely identifies a specific disk file. This list consists of the directory path plus the filename and extension.

Generic name
An abbreviation fora physical name. This abbreviation is usually three characters.

Get
To transfer a save file from a device into core using a bootstrap program or the monitor.

GLOB
A program which reads multiple binary program files and generates an alphabetic cross-referenced list of all the global symbols encountered.

Global request
A request to the LOADER to link a global symbol to a program.

Global symbol
Any symbol accessible to other programs.

GRIPE
A program that reads text from the user and records it in a disk file for later analysis by the operations staff.

Group
A contiguous set of disk clusters allocated as a single unit of storage and described by a single retrieval pointer.

High segment
The segment of the user's core which generally contains pure code and which can be shared by other jobs; it is usually write-protected.

Home block
The block written twice on every unit which identifies the file structure the unit belongs to and its position on the file structure. This block specifies all the parameters of the file structure along with the location of the MFD. The home block appears in the HOME. SYS file.

HOME.SYS
The file that contains a number of special blocks for system use. These blocks are the home blocks, the BAT blocks, the ISW blocks, and block zero.

Idle segment
A sharable high segment which users in core are not using; however, at least one swapped-out user is using it else it would be a dormant segment.

Idle time
The percent of uptime in which no job wanted to run, i.e., all jobs were HALTed or waiting for some external action such as I/O.

Immediate mode addressing
The process through which the right half of the word gives the operand and not the address.

Impure code
The code which is modified during the course of a run, e.g., data tables.

Indirect address
An address in a computer instruction which indicates a location where the address of the referenced operand is to be found.

INITIA
A program for performing standard system initialization for a particular terminal. It is used to initiate specific programs, such as the spooling programs, on the designated terminal.

Initialize
To set counters, switches, or addresses to zero or other starting values either at the beginning of or at perscribed points in a computer routine.

Interjob dependency
The technique by which a Batch job is kept from running until after the running of another job. The first job is dependent on the second job.

Interleaving
To increase effective memory speed by configuring the memory addressing so that adjacent addresses reference alternate asynchronous memories.

Internal symbol
A symbol which generates a global definition which is used to satisfy all global requests for that symbol.

Interrupt
A signal which when activated causes a transfer of control to a specific location in memory thereby breaking the normal operation of the routine being executed. An interrupt is caused by an external event such as a done condition in a peripheral. It is distinguished from a trap which is caused by the execution of a processor instruction.

ISW block
A block written by the refresher which contains the bit map for the initial storage allocation table for swapping. Any bad regions are marked as already in use. The ISW block appears in the HOME.SYS file.

Job
The entire sequence of steps, from beginning to end, that the user initiates from his interactive terminal or card deck or that the operator initiates from his operator's console.

Job Data Area
The first 140 octal locations of a user's core area. This area provides storage for items used by both the monitor and the user program.

Job search list
See File Structure Search List.

Job site
The location at which jobs are run. Also called program site.

Job step
A serial or parallel sequence of processes invoked by a user to perform an operation.

Jump
A departure from the normal sequence of executing instructions.

Label
A symbolic name used to identify a statement of a program.

Latency
(1) The time from initiation of a transfer operation to the beginning of actual transfer; i.e., verification plus search time.
(2) The delay while waiting for a rotating memory to reach a given location as desired by the user. The average latency is one half the revolution time.

LIBOL
The COBOL object time system.

Library search mode
The mode in which a program is loaded only if one or more of its declared entry symbols satisfies an undefined global request. LIB40 is scanned in this mode so as to load only programs that the user needs.

LIB40
The standard DEC-supplied library of the FORTRAN object time system and math routines. This library resides on device SYS.

Line
A string of characters terminated with a vertical tab, form feed, or line feed. The terminator belongs to the line that it terminates.

Load
To produce a core image file from a relocatable binary file (.REL) using the LOADER program. This operation is not to be confused with the GET operation: with the GET operation a core image file has already been produced.

LOADER
A program that provides automatic loading and relocation of MACRO, FORTRAN, and COBOL generated binary programs, produces an optional storage map, and performs loading and library searching. Also, the program loads and links relocatable binary programs generated by MACRO, COBOL, and FORTRAN and generates a symbol table in core for execution under DDT.

Local peripherals
The I/O devices and other data processing equipment, excluding the central processor, located at the central site.

Local symbol
A symbol used only within the program in which it is defined (all non-global symbols). It is not accessible to other programs even though the programs are loaded together.

Locked job
A job in core that is never a candidate for swapping or shuffling.

Logical device name
An alphanumeric name chosen by the user to represent a physical device. This name can be used synonymously with the physical device name in all references to the device. Logical device names allow device independence in that the most convenient physical device can then be associated with the logical name at run time.

LOGIN
The program by which the users gain access to the computing system.

LOOKFL
A program for typing the characteristics of a single disk file, such as creation date and number of words written, on the terminal.

Lost time
The percent of uptime that the null job was running, but at least on other job wanted to run (was not waiting for a device) but could not because one of the following was true:

a. the job was being swapped out.
b. the job was being swapped in.
c. the job was on disk waiting to be swapped in.
d. the job was momentarily stopped so devices could become inactive in order to shuffle job in core.

Low segment
The segment of core containing the job data area and I/O buffers. This area is unique and accessible to the user and is often used to contain the user's program. If the user is working with a shared program, this area contains data tables, etc.

MAINT.SYS
The area of the disk reserved for maintenance use only.

Macro
An instruction in a source language which is equivalent to a specified sequence of machine instructions.

Mask
(1) A combination of bits that is used to control the retention or elimination of portions of any word, character, or byte in memory.
(2) On half-duplex circuits, the characters typed on the terminal to make the password unreadable.

Master file directory
The file created at refresh time which contains the name of all user file directories including itself. Referred to as the MFD.

Master slave system
A specific multiprocessing system involving two processors where one processor has a more important role than the other.

Memory protection
A scheme for preventing access to certain areas of storage for purposes of reading or writing.

Mnemonic symbol
A symbolic representation for a computer instruction.

MONEY
A program for reading the system's time accounting file and assigning a monetary charge for each user according to the time and resources that he has used on the system.

MONGEN time
The time at which the monitor software configuration is being defined or changed. The monitor must then be reloaded with LOADER.

Monitor
The collection of programs which schedules and controls the operation of user and system programs, performs overlapped I/O, provides context switching, and allocates resources so that the computer's time is efficiently used.

Mount Count
The count of the number of jobs which have a file structure in their active or passive search lists.

Mounting a device
To request assignment of an I/O device via the operator.

Mounting a file structure
The process of adding a file structure to one's search list. If the file structure is not already defined and mounted, this is requested of the operator.

Multiprocessing
Simultaneous execution of two or more computer programs by a computer.

Multiprocessing system
A system with two or more central processors sharing some or all of the hardware resources, such as, disks memories, and or monitors.

Multiprogramming
A technique that allows scheduling in such a way that more than one job is in an executable state at any one time.

Named file
A named ordered collection of 36-bit words (instructions and or data) whose length is not restricted by size or core.

Nesting
To include a routine or block of data within a larger routine or block of data.

Non-directory device
A device such as a magnetic tape or paper tape which does not contain a file describing the layout of stored data.

No-op
An instruction that specifically instructs the computer to do nothing. The next instruction in sequence is then executed.

Non-sharable segment
A segment for which each user has his own copy. This segment can be created by a CORE or REMAP UUO or initialized from a file.

Object time system
The routines for a particular language which support the compiled code. Usually includes I/O and trap-handling routines.

Offset
The number of locations toward zero a program must be moved before it can be executed.

OMOUNT
A program for operator interfacing for handling requests concerning removable media.

ONCE ONLY time
The time at which the operator can change a number of monitor parameters when the monitor is started up.

One's complement
A complement formed by setting each bit in a binary number to the opposite state.

Operand
The symbolic addresses of the data to be accessed when an instruction is executed, or the input data or arguments of a pseudo-op or macro instruction.

Overlay
The technique of repeatedly using the same blocks of internal storage during different stages of a program. When one routine is nó longer needed in storage, another routine can replace all or part of it.

Pack ID
A 6-character SIXBIT name or number used to uniquely identify a disk pack.

Page
(1) Any number of lines terminated with a form feed character.
(2) The smallest allocatable unit of core storage.

Parity bit
A binary digit appended to an array of bits to make the sum of all the bits always odd or always even.

Parity check
A check that tests whether the number of ones or zeros in an array of binary digits is odd or even.

Passive search list
An unordered list of the file structures which have been in the job's active search list and have never been dismounted. Device DSK is not defined by this list.

Path
See directory path.

Peripheral equipment
Any unit of equipment, distinct from the central processing unit, which can provide the system with outside communication.

Physical unit name
The SIXBIT name consisting of 3 to 6 characters that is associated with each unit. Examples: FHA0, FHA1, DPA0, DPA7, LPT, DTA3.

PIP
The Peripheral Interchange Program which transfers data files from one standard I/O device to another and performs simple editing and magnetic tape control functions.

PLEASE
A program that provides the user with two-way communication with the operator.

Pointer
The location containing an address rather than data which is used in indirect addressing.

Pool
One or more logically complete file structures that provide file storage for the users and that require no special action on the part of the user.

Position operation
The operation of moving the read-write heads of a disk to the proper cylinder prior to a data transfer. This operation requires the control for several microseconds to initiate activity, but does not require the channel or memory system.

Prime RIB
The first retrieval information block (RIB) of a file. This block contains all of the user arguments.

Privileged program
(1) Any program running under project number 1, programmer number 2.
(2) A monitor support program executed by a monitor command and, therefore, has the JACCT (job status bit) set, for example, LOGOUT.

Priority interrupt
The interrupt that usurps control of the computer program or system and jumps to an interrupt service routine if its priority is higher than the interrupt currently being serviced, if any.

Process
A collection of segments that perform a particular task. A hardware state is associated with a process: a virtual memory, a processor, a stack, etc.

Program break
The length of a program; the first location not used by a program (before relocation).

Program counter (PC)
A register that, at the beginning of each instruction, normally contains an address one greater than the location of the current instruction.

Programmed operators
Instructions which, instead of doing computation, cause a jump into the monitor system or the user area at a predetermined point. The monitor interprets these entries as commands from the user program to perform specified operations.

Program origin
The location assigned by the LOADER to relocatable zero of a program.

Project-programmer number
Two octal quantities, separated by commas, which, when considered as a unit, identify the user and his file storage area on a file structure.

Protected location
A storage location reserved for special purposes in which data cannot be stored without undergoing a screening procedure to establish suitability for storage therein.

Protection address
The maximum relative address that the user can reference.

Pseudo-op
An operation that is not part of the computer's operation repertoire as realized by hardware; hence, an extension of the set of machine operations. In MACRO, pseudo-ops are directions for assembly operations.

Public disk pack
A disk pack belonging to the storage pool and whose storage is available to all users.

Pure code
Code which is never modified in the process of execution. Therefore, it is possible to let many users share the same copy of a program.

Pushdown list
A list that is constructed and maintained so that the item to be retrieved is the most recently stored item in the list, i.e., last in, first out.

QMANGR
The Batch queue manager. QMANGR is called by BATCON to schedule jobs by computing and dynamically revising job priorities.

Quantum time
The run time given to each job when it is assigned to run.

QUE
The system-wide name defining the location of the spooling and operator work-request queues.

Queue
(1) A list of jobs to be scheduled or run according to system, operator, or user-assigned priorities. Examples: Batch input queue, spooling queues, monitor scheduling queues.

(2) The system program that allows users to add, delete, list, or modify queue entries in the various system queues.

QUOLST
A program that prints the user's quotas for each file structure in his search list and the number of free blocks available in each file structure.

QUOTA.SYS
The file that contains a list of users and their quotas for the private file structure on which the file resides.

Random access
A process in which the access time is effectively independent of the location of the data.

REACT
A program for maintaining administrative control files. It can be used to create, modify, delete or list entries in a file.

Read
To open a file for input.

Record
A collection of related items of data treated as a unit.

Reentrant program
A two-segment program composed of a sharable and non-sharable segment.

Reformat
To write new headers on a disk pack using the D50B diagnostic program.

Refresh
To remove all files from a file structure and to build the initial set of files based on information in the HOM block.

Relative address
The address before hardware or software relocation is added.

Relocate
To move a routine from one portion of storage to another and to adjust the necessary address references so that the routine can be executed in its new location.

Relocation address
The absolute core address of the first location in the program segment.

Relocation constant
The number added by the LOADER to every relocatable reference within a program. The relocation constant is the relocated break of the previous program.

Remote Batch
A feature of the computing system that allows data I/O and job control of Batch processing from a distant terminal over a synchronous communication link.

Remote peripherals
The I/O devices and other data processing equipment, except the central processor, located at the site of the remote Batch terminal.

Removing a file structure
The process of physically removing a file structure from the system. This is requested with the REMOVE switch in the DISMOUNT command string and requires the operator's approval.

Response time
The time between the generation of an inquiry and the receipt of an response.

Return
(1) The set of instructions at the end of a subroutine that permits control to return to the proper point in the main program.
(2) The point in the main program to which control is returned.

Run
To transfer a save file from a device into core and to begin execution.

RUNOFF
A program that facilitates the preparation of typed or printed manuscripts by performing formatting, case shifting, line justification, page numbering, titling, and indexing.

SAT.SYS
The Storage Allocation Table file which contains a bit for each cluster in the file structure. Clusters which are free are indicated by zero and clusters which are bad, allocated and non-existent are indicated by one.

Save
To produce a save file from a core image using a bootstrap program or the monitor. This operation is the opposite of the GET operation.

SCRIPT
A program that sends predetermined sequences of characters over multiple pseudo-TTYs in order to simulate a load on the system for analysis.

Search
The Controller reads sector headers to find the correct sector. The second step in the transfer operation.

Sector
A physical portion of a mass storage device.

Segment
A logical collection of data, either program data or code, that is the building block of a program. The monitor keeps a segment in core and/or on the swapping device.

Segment Resident Block
A block that contains all the information that the monitor requires for a particular segment.

SETSRC
A program that allows the user to list or change the search list that is automatically set up for him at job initialization time.

SFD
A directory pointed to by a UFD or a higher-level SFD. These directories exist as files under the UFD.

Sharable segment
A segment which can be used by several users at a time.

Shared code
Pure code residing in the high segment of user's core.

Single access
The status of a file structure that allows only one particular job to access the file structure. This job is the one whose project number matches the project number of the owner of the file structure.

Skip
An instruction that causes control to bypass one instruction and proceed to the next instruction.

Spooling
The technique by which output to slow-speed devices is placed into queues to await transmission; this allows more efficient use of the particular device, core memory, and the central processor unit.

Static dump
A dump that is performed at a particular point in time with respect to a machine run, frequently at the end of a run.

STRLST.SYS
The administrative file that describes each file structure in the system. This file is used by the MOUNT command only.

Sub-directory
A continued SFD.

Supersede
To open a file for writing, write the file and close the file when an older copy of the same name already exists. Only one user at a time may supersede a given file at any one time. The older copy of the file is deleted when all users are finished reading it.

Super-cluster
A contiguous set of one or more clusters introduced to compress the file pointer for large units into 18 bits. See compressed file pointer.

Swapping
The movement by the monitor of user programs between core and secondary storage.

Swapping class
The classes of swapping units divided according to speed. Class 0 contains the fastest swapping units.

Swapping device
Secondary storage that is suitable for swapping, usually a high-speed disk or drum.

SWAP.SYS
The file containing the swapping area on a file strucutre.

Symbolic address
An address used to specify a storage location in the context of a particular program. Symbolic addresses must then be translated into absolute addresses by the assembler.

Symbol table
A table which contains all defined symbols and the binary value assigned to each symbol.

SYS
A system-wide logical name for the system library. This is the area where the standard programs of the system are maintained.

SYSDPY
A variation of the SYSTAT program which runs on a keyboard display at up to 2400 baud.

SYS search list
The file structure search list defined at ONCE-ONLY time for device SYS.

SYSTAT
A program that displays on the user's terminal the status of the system at any time.

TECO
A sophisticated text editor and corrector program that allows simple editing requests, character string searches, complex program editing, command repetition, and text block movement. TECO editing is performed on files recorded in ASCII characters.

TENDMP
A utility program used to save and restore core images on DECtape or magnetic tape. It operates only in executive mode.

Total user core
The amount of physical core which can be used for locked and unlocked jobs.

Track
The portion of a moving storage medium, such as disk, drum, or tape, that is accessible to a given reading head position.

Transfer operation
The hardware operation of connecting a channel to a controller and a controller to a unit for passing data between the memory and the unit. The transfer operation involves verification, search, and actual transfer.

Trap
An unprogrammed conditional jump to a known location, automatically activated by a side effect of executing a processor instruction. The location from which the jump occurred is then recorded. It is distinguished from an interrupt which is caused by an external event.

Two's complement
A number used to represent the negative of a given value. This number is obtained by alternating the bit configuration of each bit in the binary number and adding one to the result.

UFD
A file whose entries are the names of files existing in a given project-programmer number area within a file structure.

UMOUNT
A program for user interfacing for the handling of requests concerning removable media.

Unconditional transfer
An instruction which transfers control to a specified location.

Unit
The smallest portion of a device that can be positioned independently from all other units. Several examples of units are: a disk, a disk pack, and a drum.

Update
To open a file for reading and writing simultaneously on the same software channel, rewrite one or more blocks in place, and close the file. Only one user at a time may update a given file.

User
A person who utilizes the facilities of the DECsystem-10.

User file directory
See UFD.

User I/O mode
The central processor mode that allows privileged user programs to be run with automatic protection and relocation in effect, as well as the normal execution of all defined operation codes.

User library
Any user file containing one or more programs of which some or all can be loaded in library search mode.

User mode
A hardware-defined state during which instructions are executed normally except for all I/O and HALT instructions which cause immediate jumps to the monitor. This makes it possible to prevent the user from interferring with other users or with the operation of the monitor. Memory protection and location are in effect so that the user can modify only his area of core.

User program
All of the code running under control of the monitor in an addressing space of its own.

Verification
The controller reads sector headers to see if the mechanical parts of the system have correctly positioned the arm. The first step in the transfer operation.

Vestigial job data area
The first 10 locations of the high segment used to contain data for initializing certain locations in the job data area.

Virtual core
The size of the job, both low and high segments.

Wildcard construction
A technique used to designate a group of files without enumerating each file separately. The filename, extension, or project-programmer number in a file specification can be replaced totally with an asterisk or partially with a question mark to represent the group of files desired.

Word
An ordered set of bits which occupies one storage location and is treated by the computer circuits as a unit. The word length of the DECsystem-10 is 36 bits.

Zero compression
The technique of compressing a core image by eliminating consecutive blocks of zeros.

APPENDIX A DECsystem-10 HARDWARE

DECsystem-10 is the name for the family of DEC's large computing systems. Each of the five systems in the DECsystem-10 range is centered around one or two PDP-10 central processors. The systems are distinguished from each other by their range of performance, which is achieved by adding more hardware. The additional hardware that increases performance in the expansion from a small to a larger system includes: swapping devices, central processors, core memories, and peripheral equipment, including data communications systems. The systems have no fixed hardware boundary because an individual system can be expanded to any size. No software changes are required in expanding an individual system; all configurations of the DECsystem-10 use the same operating system for all applications.

A.1 DECsystem-1040

The 1040 is the smallest of the five systems. The typical configuration of this system has a KA10 central processor, 32 to 64K high-speed ME10 core memories, the RP02G disk system with up to two disk packs, the TM10G magnetic tape system with up to two drives, and low-speed peripheral equipment including a CR10F card reader, an LP10A line printer, and local DC10 lines. This is an excellent system for the scientific research lab where multiple real-time tasks and general computing are required, and also for small colleges where there is a need for handling administrative, student, and faculty workloads simultaneously. The system is easily expandable with most equipment on the DECsystem-10 Equipment List.

A.2 DECsystem-1050

The 1050 is a full capability, medium power system. The addition of a high-speed RM10G swapping drum system substantially increases the number of simultaneous users on the system. Other components of this system include: the KA10 central processor, 64 to 96K high-speed ME10 core memories, the RP02G disk system with up to four disk packs, the TM10G magnetic tape system, the CR10D card reader, the LP10C line printer, and 32 local lines in either the DC10 or DC68A communications system. The 1050 is well-suited for the educational and scientific environments because it has the capability of running ALGOL, BASIC, COBOL, and FORTRAN compilers concurrently on a configuration that is economically priced and easy to learn and use. Business data processing areas find that with the 1050, COBOL program preparation is enhanced by interactive editing and debugging via local or remote terminals.

A.3 DECsystem-1055

The 1055 is the dual processor equivalent of DECsystem-1050 with fast execution of compute-bound jobs because of the addition of the second processor. This system has two parallel KA10 processors connected with one operating system in order to double the computing power of the 1050 and at the same time to maintain the same interface between the user and the computing system. This approach of co-equal processors gives the user increased computing capacity when processing power is in heavy demand under multi-task loads. In addition to the two KA10 processors, the typical 1055 has 80K of ME10 core memories, with one MX10 memory port multiplexer, one RM10G drum system, one RP03G disk system with up to eight disk packs, one TU40G, 120KC magnetic tape system, one CR10 card reader, the LP10C line printer, and 32 local lines, either a DC10 system or a DC68A system.

A.4 DECsystem-1070

The 1070 is a large-scale computing system with more than twice the central processor speed of the DECsystem-1050 because of the KI10 central processor. This processor has hardware memory paging, double-precision floating-point arithmetic, instruction look-ahead, and virtual memory capability. In addition to the

KI10 processor, the typical 1070 comprises at least 96K (480K bytes) of ME10 core memory, 690K words (4.1 million characters) of RM10G high-speed drum storage, an RP03G disk system of four disk drives with a total of 41.6 million words (249.6 million characters) of storage, TU40G magnetic tape system with three 120KC drives, a 1200 character-per-minute CR10E card reader, a 1000 line-per-minute LP10C line printer and a communication system capable of 128 lines (either DC10 or DC68A). With the increased memory size, the high performance peripheral systems, and the large file system, the 1070 is configured for maximum support of remote batch capabilities through the synchronous communication equipment. Multiple remote stations have simultaneous access to the DECsystem-1070, with each capable of concentrating up to 16 terminals to its computer.

A.5 DECsystem-1077

The 1077 is the dual-processor 1070 with fast execution of computing loads because of the second KI10 central processor. In addition, this system typically has 128K (640K bytes) of core memory, 690K words (4.1 million characters) of RM10G drum storage, a RP03G disk system with four disk drives for a total of 41.6 million words (249.6 million characters) of storage, a TU40G magnetic tape system with four 120KC drives, a 1000 line-per-minute LP10C line printer, a 1200 character-per-minute CR10E card reader, and a DC10 or DC68A communication system capable of 128 lines. In expanding to the 1077 from a smaller system, the user notices increased computing power, but he does not need to change his programs or learn a new command language or operating system.

A.6 PROCESSOR — KA10

The KA10 arithmetic processor is the processing unit for the three smallest DECsystem-10 machines. Its standard I/O devices are: a. a 300 character-per-second photoelectric paper-tape reader, b. a 50 character-per-second paper-tape punch, c. an operator's console that provides the operator with information and intervention capabilities when desired, and d. a standard Model 35KSR console teleprinter operating at 10 characters-per-second (considered as part of the operator's console). The 36-bit instruction word format of the KA10 provides 512 operation codes, of which 366 are hardwired. The remainder are programmed operators or are reserved for future use.

The fast registers, KM10, are sixteen 36-bit integrated circuit registers used as multiple accumulators, index registers, or memory locations. These registers have an access time of 200 ns and when used as memory locations can double the execution speed of a program. The dual memory protection and relocation registers, KT10A, allow the software to define two areas for each user and to protect the remaining of core from these users.

The priority interrupt system of the central processor has seven levels of interrupts for the devices attached to the I/O bus. The entire priority interrupt system is programmable. With software, any number of devices can be attached to any level, individual levels or the entire priority interrupt system can be deactivated and later reactivated, and interrupts can be requested on any level. With the executive control logic, the KA10 operates in one of three modes: a. executive mode, which allows all instructions to be executed and suppresses relocation. b. user mode, in which some instructions are not allowed (i.e., I/O instructions) and relocation and protection are in effect, and c. user I/O mode, where all instructions are valid but relocation and protection are still in effect.

A.7 PROCESSOR — KI10

The KI10 central processor used with the larger DECsystem-10 machines is nearly twice as fast as the KA10 processor. This increase in speed results from the use of different architecture, faster circuits, a more complex adder, improved algorithms, and lookahead instruction logic, which obtains the next instruction during the execution of the current instruction.

Core memory is managed by the paging system of the KI10. This system allows the user program to access an effective address space of up to 256K words. This space is segmented into 512_{10} pages of 512_{10} contiguous words each. These pages do not have to be contiguous in the physical core memory.

The KI10 processor provides memory address mapping from a program's effective address space to the physical address space by substitution of the most significant bits of the effective address. This mapping provides access to the entire physical memory space, which is 16 times larger than the effective address space. (The program's effective address space is 256K (18 bits); the physical address space is 4096K (22 bits)). Memory mapping takes place using a page table as follows: the most significant nine bits of the effective address, the page number, is used as an index into the appropriate page table. The effective page number is then replaced by the information located in the page table entry. This information is a physical page number of 13 bits. These 13 bits are concatenated with the least significant 9 bits of the effective address, the word address within the page, in order to form the 22-bit physical address. More core is then able to be addressed when providing a physical address space much larger than the effective address space. This gives programs the ability to access 4 million words.

Eight instructions for double-precision floating-point arithmetic and three instructions for converting between fixed-point and floating-point formats are in the KI10 instruction repertoire. The double-precision word format gives precision of 1 part in 4.6 x 10^{18} and an exponent to the power of 256.

The KI10 processor provides measures for handling arithmetic overflow and underflow conditions, pushdown list overflow conditions, and page failure conditions directly by the execution of programmed trap instructions instead of resorting to a program interrupt system. The trap instruction is executed in the same address space as the instruction that caused the trap. Therefore, user programs can handle their own traps by directing the monitor to place a jump to a user routine in the trap location.

The maximum uninterruptable interval on the priority interrupt system is 10μs. The I/O bus cycle time of the KI10 processor is 2.7μs. Interrupt response is enhanced by the four blocks of general-purpose registers. Each block contains 16 registers that facilitate both rapid context switching between programs and interrupt handling.

The KI10 operates in one of two modes, user mode and exec mode. Each of these modes have two submodes: a. public mode and concealed mode in user mode, and b. supervisor mode and kernel mode in exec mode.

User programs operate in user mode. In this mode, the program can access up to 256K words. All instructions are legal except those that interfere with other users or the integrity of the system. A program in public mode can transfer to a program in concealed mode only by transferring to locations that have ENTRY instructions. A program in concealed mode can read, write (if allowed), execute, and transfer to any location designated as public. Concealed mode allows the loading of proprietary software with a user program and data, but prevents the user program from changing or copying the software. This provides direct interaction between the user and the proprietary software with virtually no overhead.

The operating system operates in exec mode. The smaller part of the operating system operates in kernel mode and performs both I/O for the system and any functions that effect all users of the system. The larger part of the operating system operates in supervisor mode and performs general management of the system and the functions that effect only one user at a time.

A.8 CORE MEMORIES

The ME10 core memory contains 16,384 words with a read access time of 600 nanoseconds and a full cycle time of one microsecond. Up to 16 memory modules can be connected to provide 256K of core storage. Each module can contain up to four ports. This memory features both two-and four-way interleaving with switches on each memory module. It is specifically built for the KI10 processor in that it can recognize the 22-bit address space. It also takes advantage of the overlap memory control of the KI10, which results in a 20% increase in speed.

The MD10G mass memory system consists of 64K MD10 core memory and a MD10E including cables. The basic unit of the MD10 memory has 65,536 words of storage at 36 bits per word. The unit has an access time of 830 ns, a cycle time of 1.8μs, and two-or four-way interleaving between cabnets. This memory is equipped with four access ports for connection to the processor and data channels. The MD10E core memory expansion module expands the MD10E up to 128K in increments of 32,768 words.

The MD10H mass memory system consists of 128K MD10 core memories and three MD10Es including cables.

A.9 DRUM SYSTEM

The RM10G drum system consists of a DF10 data channel, a RC10 fixed-head drum control, and a RM10B fixed-head drum. The DF10 controls the transfer of data between a device controller and one port in memory. Up to eight controllers or special devices can be connected to the DF10, providing one data path to core memory. In other words, one device can be transferring data, and other devices on the DF10 must wait until the device has completed the data transfer. The rate of transfer is determined by the speed of the device using the DF10. The RC10 controls up to four RM10B drums. It connects to the processor via the I/O bus for control and status information. Under program control, it establishes a data path between the drum and a core memory port via the

DF10. The RM10B provides 345,000 36-bit words for fast-access storage available for swapping, data storage, and program libraries. It has an average latency time of 8.5 ms and an average transfer time of 4.5 μs per 36-bit word (or about 10.2 ms and 5.4 μs respectively when operating with 50 Hz power). Due to its speed, the drum should be connected to the highest priority memory port via the DF10.

A.10 DISK SYSTEMS

The RP02G disk system consists of the DF10 data channel, a RP10 disk control, and two RP02 disk pack drives. The RP10 disk control can provide control of up to eight RP02 disk pack drives. It connects to a DF10 data channel and the I/O bus. The RP02 disk drive provides storage for up to 5,120,000 36-bit words on interchangeable disk packs. The average access time is 47.5 ms, which includes 12.5 ms average rotational latency, and the transfer rate is 15 μs per word.

The RP03G disk system includes a DF10 data channel, a RP10C disk control, and four RP03 disk pack drives. The RP10C can control up to eight RP02 or RP03 (or a combination of the two) disk pack drives. The RP03 has a total of 400 cylinders that give twice the storage capacity of the RP02.The average access time is 41.5 ms including the 12.5 ms average rotational latency, and the transfer rate is identical to the RP02.

The maximum disk system storage capacity is: up to four controllers, each with eight drives, giving a total of 327,680,000 words, or in excess of 1.966 x 10^9 characters of on-line storage.

A.11 MAGNETIC TAPE SYSTEMS

The TD10G DECtape system consists of a TD10 DECtape controller and a TU56 DECtape transport. The TD10 controls either four TU56 dual DECtape transports or eight TU55 DECtape transports. Data is transferred between the TD10 and the central processor over the I/O bus at the average rate of one 36-bit word every 400 μs. The TU56 transport reads and writes magnetic tape at 15K characters per second. The tapes are 3-3/4 in. in diameter, 260 ft. long, and 3/4 in. wide. Each tape has a directory providing random access to user files. The tape units are bidirectional and redundantly recorded, resulting in greater maintainability and reliability.

The TM10G 36KC magnetic tape system has a TM10A magnetic tape control and either two TU10 or two TU20 magnetic tape units. The TM10A controls the operation of up to eight tape transports and provides a data path from the tape transport to the central processor via the I/O bus. The data transfer rate is determined by the speed and density of the drive being controlled. The TU10 magnetic tape unit reads and writes 9-channel (TU10E) or 7-channel (TU10F) industry standard tape at 45 in. per second and a density of 200, 556, and 800 bits per inch (TU10E) or bits per second (TU10F). The TU20A magnetic tape unit reads and writes 9-channel USASI standard magnetic tape at a rate of 45 in. per second and with a density of 800 bits per inch. The TM10 controller assembles four 8-bit characters per 36-bit word transfer. The TU20B magnetic tape unit reads and writes 7-channel industry standard magnetic tape at the rate of 45 in. per second and with densities of 200, 556, and 800 bits per inch. The TM10 controller assembles six 6-bit characters per 36-bit word for transfer.

The TU40G 120KC magnetic tape system includes a DF10 data channel, a TM10B magnetic tape control, and two TU40 magnetic tape units. The DF10 controls the transfer of data between eight device controllers and one port of core memory. The TM10B controls up to eight tape transports. This control uses the I/O bus to receive information from and to provide status to the processor. It establishes a data path from the tape transport to core memory via the DF10. The transfer rate of the control is determined by the speed and density of the tape transport performing the transfer. The TU40 reads and writes 9-channel USASI standard magnetic tape at 150 in. per second and a density of 200, 556, and 800 bits per inch. The TU41 reads and writes 7-channel industry standard tape at 150 in. per second and a density of 200, 556, and 800 bits per inch.

A.12 INPUT/OUTPUT DEVICES

A.12.1 Card Readers

The card readers offered with the DECsystem-10 have insignificant card wear, high tolerance to damaged cards and are virtually jamproof. The life of an individual card has been proven to be in excess of 1000 passes. These readers are designed to permit the operator to load and unload cards while the reader is operating.

The CR10D card reader is a table-top model that reads 80-column EIA standard cards at 1000 cards per minute. The capacity of the card hopper is 1000 cards. The card reader controller connects to the BA10 hard copy controller.

The CR10E console model card reader inputs 80-column EIA standard cards at 1200 cards per minute. The maximum number of cards held by the input and output hoppers is 2250 cards. The controller is mounted in the BA10 hard copy controller cabinet.

The CR10F card reader is a table-top model and reads 80-column EIA standard cards at the rate of 300 cards per minute. The hopper of the CR10F holds 600 cards. The controller connects to the BA10 hard copy controller.

A.12.2 Card Punch

The CP10A card punch punches cards at the rate of either 200 cards per minute when punching all 80 columns or 365 cards per minute when punching only the first 16 columns. The card hopper and stacker capacities are 1000 cards. The card punch controller is mounted in the BA10 hard copy controller cabinet.

A.12.3 Line Printers

The 64-character LP10A line printer prints 300 lines per minute and 132 columns per line. The printable character set is composed up upper-case characters, numbers, and special characters. The line printer is connected to the I/O bus via a controller mounted in the BA10 hard copy controller.

The 64-character LP10C line printer prints 1000 lines per minute and 132 columns per line. The printable character set is the same as the LP10A character set. The line printer is connected to the I/O bus with the BA10 hard copy controller.

A.12.4 Plotters

The XY plotter control is the interface for CalComp 500 and 600 series digital incremental plotters. It is normally mounted in the BA10 hard copy controller, but can be mounted in the TD10 DECtape controller cabinet.

A.12.4.1 XY10A CalComp Plotter Model 565—The XY10A plotter is interfaced to the I/O bus via a controller mounted in the BA10. This plotter has the following specifications:

Step Size	Steps Minute	Width of paper
0.01 in.	18,000	12 in.
0.005 in.	18,000	12 in.
0.1 mm	18,000	12 in.

A.12.4.2 XY10B CalComp Plotter Model 563— The XY10B plotter is interfaced to the I/O bus via a controller mounted in the BA10. The plotter has the following specifications:

Step Size	Steps Minute	Width of paper
0.01 in.	12,000	31 in.
0.005 in.	18,000	31 in.
0.1 mm	18,000	31 in.

A.12.5 BA10 Hard Copy Control

The BA10 control cabinet contains the controllers for the card readers, card punch, line printers, and plotters. It has the power supplies and fans necessary to support the controllers.

A.13 TELETYPES AND TERMINALS

The Teletypes and Terminals used on the DC10 and the DC68A are similar except for different cables and interface connectors.

A.13.1 Local DC10 Use

The LT33A teleprinter is the 33TS machine (33KSR, friction feed).

The LT33B teleprinter is the 33TY machine (33ASR, sprocket feed, automatic reader control XON/XOFF feature).

The LT35A teleprinter is the VSL312HF machine (35KSR, sprocket feed).

A.13.2 Local DC68A Use

The LT33C teleprinter is the 33TS machine (33KSR, friction feed).

The LT33H teleprinter is the 33TY machine (33ASR, sprocket feed, automatic reader control XON/XOFF feature).

The LT35C teleprinter is the VSL312HF machine (35KSR, sprocket feed).

A.13.3 CRT displays

The VT06 alphanumeric terminal is a CRT display terminal capable of transmitting data locally or over standard phone lines using data sets conforming to the RS-232-C standard. The VT06 can functionally be interchanged with a teleprinter. In addition, the VT06 can be used for display-oriented operations by utilizing the cursor-control features. It has 25 lines of 72 characters each and operates asychronously full-or half-duplex at a variety of baud rates up to 2400 baud, selectable by a switch on the rear panel.

The VT05 alphanumeric terminal is a CRT display terminal capable of full-and half-duplex data transmission at rates up to 300 baud. Alphanumerics can be superimposed over a video image derived from closed circuit TV or video tape.

A.14 DATA COMMUNICATIONS SYSTEMS

The data communication equipment includes two systems for asychronous communications (hardwired and programmable), two systems for synchronous communications (low capacity and high capacity) and a remote batch terminal.

A.14.1 DC10 Data Line Scanner

The DC10 hardwired data line scanner interfaces asychronous communications lines to the processor via the I/O bus. The DC10A control unit is the basic unit and contains the I/O interface and control logic. This unit provides on-line servicing of up to 64 local communication lines. It accomodates any device that uses eight-or five-level serial teletype code. Standard system software supports interactive ASCII terminals at speeds up to 2400 baud. For some special communication applications, the DC10 can operate at higher speeds. Full-duplex with local copy and half-duplex data modes are available on each line serviced.

The DC10B is an eight-line group unit connected to the DC10A and provides an interface for up to eight local lines. It can be used in full-duplex or full-duplex with local copy operation. To provide carrier detection or data set status control, the DC10E is required.

The DC10C eight-line telegraph relay assembly provides an interface to long distance telegraph lines using full-or half-duplex facilities.

The DC10D telegraph power supply is the standard line voltage supply used with DC10C (120 Vdc at 2 A).

The DC10E data set control provides expanded processor control of eight data sets in the DC10 system.

A.14.2 DC68A Communication System

The DC68A programmable communications system is built around the 680/I communications version of the PDP-8/I. Characters are assembled via program control, which results in a very low incremental cost per line. The DC68A is optimized for a large number of 110 baud lines, but will operate at speeds up to 300 baud. The PDP-8/I is under monitor control and transfer across the interface occurs on the character-by-character basis. The DC68A provides on-line servicing of up to 63 communication lines. Terminals can be local or remote through data sets. The standard configuration includes one DA10 interface, one PDP-8/I-D computer (4K of memory with MP8/I parity option, and a Model 33ASR teleprinter), one DL8/I serial line adapter, one DC08A serial line multiplexor, and three clocks for line frequency operations at 110, 150, or 300 baud rates. Additional options mentioned in this section are required for implementing a specific number of local or data sets.

The M750 dual serial line adapter implements two full-duplex channels in the basic communication system. One unit is required for every two local or data set lines. The DC08B local line panel accommodates up to 48 local terminals suitable for direct 680/I connection. The DC08F modem interface and control multiplexor accommodates up to 32 dual modem control units to handle up to 64 asynchronous lines. The DC08G dual modem control unit implements two modem control units in the DC08F. It includes 25 ft. cables with modem connector DB-25D.

A.14.3 DS10 Synchronous Line Unit

The DS10 synchronous line unit is an interface between the DECsystem-10 I/O bus and one full-or half-duplex voice grade synchronous modem to a remote batch terminal, high-speed display, remote job entry station, or another computer. The synchronous modem meets EIA RS-232B or C standards, such as the Bell System 201B. System software supports full-duplex operation of an DS10 at up to 9600 baud, or two DS10s at up to 4800 baud each.

A.14.4 DC75 Synchronous Communications System

The DC75 synchronous communication system is a PDP-11-based front-end system designed to efficiently handle multiple synchronous lines. The basic DC75 system includes a DL10 interface, one PDP-11/20, and a DS11 synchronous modem interface implemented for eight lines.

The DL10 is a direct memory interface between the DECsystem-10 memory and the PDP-11 communications processor. Each DL10 can connect up to four PDP-11s.

A basic DC75 system can handle eight full-duplex lines at speeds up to 4800 baud each, or four lines at 9600 baud. It can be expanded to handle 16 lines at 2400 baud by implementing additional DS11 line capability.

For applications requiring additional line capability at 4800 baud or 9600 baud, up to three additional PDP-11/DS11 combinations can be added to the DL10 interface unit. Each additional PDP-11/DS11 combination provides a line throughput capability equal to the initial system.

For special applications, the DC75 can be programmed to handle a mix of line speeds, character sizes, and message formats. The DS11 modem interface hardware has provision for 6-, 8-, or 12-bit character sizes, and these characters can be efficiently packed into DECsystem-10 memory by the DL10.

A.14.5 DC71 Remote Batch Station

The DC71 remote batch station consists of a PDP-8/I processor, an operator Teletype, a card reader, a line printer, and a synchronous interface. The DC71 connects to the DS10 or the DC75 via a full-duplex synchronous communications link. The remote batch terminal can be either a DC71A or DC71B terminal. The DC71A is configured with a 132-column line printer with a 64-character set. The DC71B is configured with a 96-character set line printer. The DC71D Teletype Concentrator package includes eight lines for connecting to the DC71A or DC71B. Another eight lines can be added by connecting the DC71E to the DC71D. Terminals can be Teletypes, VT06 or VT05 display terminals, or other teletype-compatible terminal interfaces, at speeds up to 2400 baud.