PART 2

DOS/BATCH CONCEPTS AND CAPABILITIES

PART 2

CHAPTER 1

INTRODUCTION

1.1 FUNCTIONS OF AN OPERATING SYSTEM

An operating system exists to fulfill the following functions:

1. Organize a collection of hardware into a consistent and coherent whole so that it can be used for the development and running of programs.
2. Assist users and their programs to make maximum use of the hardware with minimal programming effort by providing software to supplement and enhance the hardware facilities available.

1.2 DESIGN CRITERIA

An operating system must be designed to perform the functions specified above in accordance with the following criteria:

1. Most efficient use of resources, especially memory,
2. Maximum speed, and
3. Maximum ease of use.

These criteria vary in priority depending upon the kind of work that the operating system is to handle. For example, for an on-line real-time data acquisition application, speed is paramount. For large-scale commercial processing the main consideration is likely to be maximum throughput and efficiency; the speed at which each job is processed would not be critical.

1.3 THE ROLE OF DOS/BATCH

DOS/BATCH has been designed to manage resources efficiently and to be easy to use. It is suitable both for regular production work and, by reason of its data storage facilities and debugging aids, for program development. However, while it provides a response time fast enough for most requirements, it is not intended for use in on-line real-time data acquisition applications.

1.3.1 Batch Operation

The single-user interactive system is probably the most inefficient use of resources. A command which can be executed in a thousandth of a second may take several seconds to type.

DOS/BATCH batch facilities permit large numbers of programs or jobs to be prepared off-line on paper tape or punched cards, and then read and run in quick succession. Programs or jobs which are to be run more than once can be stored on and read from disk or tape even more efficiently. Batch operation is described in Section 2-6.3.2 and Part 4.

1.4 AREAS OF ACTIVITY

The DOS/BATCH Monitor carries out the functions described in Section 2-1.1 in three main areas of activity:

1. Two groups of services to running programs.

   a. The handling of input and output. This is one of the primary functions of a Monitor. Chapter 2-2 shows that the user is given the opportunity to request as much or as little assistance as he needs. Normally he need not consider the devices the program will actually use when it is executed.

   b. Other program services. These include loading the program, assisting in data manipulation within the program, and supplying system information. The utility routines which provide these services are described in Chapter 2-5.

2. Storing and retrieving sets of data on magnetic media. DOS/BATCH uses a system of named files; a single user can manipulate sets of data easily and conveniently, or several users may store data on one medium without conflict.

3. Management of memory. The Monitor must efficiently handle the competing demands of programs, data buffers, and Monitor routines for space in memory. In particular, the Monitor itself must be designed so that it occupies only a small part of available memory, leaving the remainder free for user programs and data buffers. The manner in which DOS/BATCH accomplishes this is described in Chapter 2-4.

The activities listed above must be performed as far as possible independently of the user and his program. The success of the Monitor's memory management activities can be measured in part by the extent to which the user is unaware of them. However, the user may wish to intervene, or at least keep himself informed in the other areas, since he must give the initial request for each Monitor action, such as the performance of input or output or the creation of a file. Therefore, the means must be provided for Monitor user communication.

Under DOS/BATCH the user can communicate with the Monitor at two levels - from within his program by means of a FORTRAN statement or MACRO program request, and through a comprehensive keyboard command language. The Monitor communicates with the user by means of predefined standard responses and error messages.

1.5 PROGRAM DEVELOPMENT

DOS/BATCH is particularly valuable as an environment in which programs may be developed. This is because of the convenience afforded by the file system, and because of the system programs which are provided to assist in this function. These facilities are described in Chapter 2-7.

1.6 SYSTEM GENERATION AND ADAPTABILITY

An operating system should be easy to set up, both initially and on a regular basis, easy to tailor to the particular configuration upon which it is used, and easy to modify. The manner in which DOS/BATCH satisfies these requirements is described in Chapter 2-8.

PART 2

CHAPTER 2

INPUT AND OUTPUT

A considerable quantity of code is inevitably required to effect input and output of data to and from a program - typically much more than is required to process it. While there are as many ways of processing as there are programs, the number of ways in which input and output can be performed is limited. Clearly, it is desirable that programs should be relieved of the need to handle the details of input and output operations. This chapter describes the way in which the DOS/BATCH Monitor manages data input and output on behalf of user programs.

2.1 DEVICES SUPPORTED

DOS/BATCH supports the following media and devices for data input and output:

1. For both input and output:
   a. Magnetic disks (of various kinds),
   b. DECtape (three-quarter inch tape on 3.9 inch diameter reels),
   c. Magnetic tape (7- and 9-track) (industry-standard half inch tape),
   d. Magnetic tape cassettes,
   e. Paper tape, and
   f. Keyboard terminals (of various kinds).
2. For input only:

   Punched cards.
3. For Output only:

   Line printers (of various capabilities).

A list of the actual devices currently supported appears in Appendix H.

2.2 OBJECTIVES

The DOS/BATCH input and output facilities have been designed to achieve the following objectives:

1. Device independence,
2. Variety of access methods,
3. Memory economy, and
4. Program efficiency.

These objectives are discussed in the following sections.

2.2.1 Device Independence

Whenever possible input and output operations should have a standard interface with the program, so that program coding is not dependent on device characteristics. Devices may then be changed without any effect on the program.

In addition, it should be possible to make device specification at run time; or, alternatively, override the program's device specification.

2.2.2 Variety of Access Methods

The two principal methods of access to data, namely sequential access and random access, should be fully supported. The transfer of both binary data and coded character strings should be possible. Data formatting and parity checking should be available if required.

2.2.3 Memory Economy

Because of the quantity of code required to effect input and output, and the large number of peripheral devices supported, the part of the Monitor that handles input and output will inevitably be large. Therefore, a modular design is essential, so that at any one time only those parts which are required need be in memory.

2.2.4 Program Efficiency

The Monitor should be able to service an input or output request while the program continues processing, and even service a second input or output request before the previous one has been completed.

2.3 DATA

Data may exist as a series of holes punched on cards or paper tape; it may be held on magnetic storage devices; it may be input by pressing the keys on a keyboard; and it may be output in the form of printed characters on paper or a screen. However, if the objectives of device independence and interchangeability are to be attained, the Monitor must not treat these forms of data differently, and the user must not be forced to think of them as intrinsically different.

2.3.1 Data Definition

The first step towards device independence under DOS/BATCH is definition of sets of data in terms of datasets and files.

2.3.1.1 Datasets

A dataset is a logical grouping of data identified by a single name or specifier. It can contain part or all of one or more physical units of data.

For example, data punched on a paper tape which is loaded into a paper tape reader may be identified by the device name PR:. When it is so identified, it is being treated as a unit and therefore constitutes a dataset.

It should be noted that the term "dataset" has no absolute or permanent meaning. A grouping of data becomes a dataset only when it is specified as a unit.

2.3.1.2 Files

One of the chief advantages of magnetic media (disk, DECtape, magnetic tape and cassettes) is their great capacity. Only rarely does a user wish to handle data in units big enough to fill such a medium. Normally he wishes to store on the medium a number of sets of data, each of which he wishes to treat as a unit. In other words, it is necessary to be able to identify several datasets on one medium.

DOS/BATCH facilities provide for attaching a name to a set of data when it is transferred to a magnetic storage medium. Such a named set of data is known as a file.

Filenames are attached to sets of data for convenience, because it is expected that the user will subsequently wish to treat the same set of data as a unit - that is, a dataset. A filename, like a device name, is a means of identifying a set of data, and when it is so used the contents of the file constitute a dataset. However, this is not to say that a file is a dataset. As mentioned previously, the term "dataset" has meaning only when a set of data (which may be a file) is actually specified. Indeed, the identification of a set of data as a file does nothing to preclude that data from being included subsequently in a larger dataset, nor, in certain cases, to preclude the specification of smaller datasets within that file.

It is possible, for example, to specify that the whole contents of a medium, including all the files held on it, be transferred to another medium. If this is done the complete contents of the medium is, for the purpose of that transfer, a single dataset.

Parts of files may also be specified; that is, datasets may be subsets of files. A library is a type of file which is specifically designed so that either the whole or previously specified parts may be specified as datasets.

2.3.1.3 Device Names

A device name specifies a device and consists of a standard two-letter mnemonic code, optionally followed by a digit, and terminated by a colon. If there is more than one unit of the same kind of device, the digit is used to differentiate between them. Examples of device names are:

DT1:	The DECtape unit numbered 1.
MT0: or MT:	The magnetic tape unit numbered 0, or the only magnetic tape unit.

The omission of the digit has the same effect as specifying 0.

2.3.1.4 Filenames and Filename Extensions

A filename consists of up to six letters and digits; with the restriction that the first character must be a letter. It may be followed by a filename extension, which consists of up to three letters or digits, separated from the filename by a period. Filenames and extensions in general are chosen by the user, but in many cases the system supplies a standard extension.

Examples of filenames:

```
INDAT1
A174.001
PROG1.MAC
PROG1.OBJ
PROG1.LDA
```

The last three examples illustrate one use of standard extensions. All three files hold the same program, PROG1, but each is in a different form. PROG1.MAC is the source program, written in the PDP-11 assembly language MACRO; PROG1.OBJ is the same program after assembly (known as an object module); and PROG1.LDA is the same program in a form that is ready to be loaded and run (a load module). These extensions are added to the filenames automatically when the files are created unless other extensions are specified by the user.

The first steps, then, towards device independence under DOS/BATCH are the concept of a dataset and the ability to define a dataset by means of a device name alone or, on devices which support files, by a device name coupled with a filename. For example, the same set of data, copied from a non-magnetic device to a magnetic device, can be specified by a device name and filename (example: DT:INDAT) as a dataset for output.

2.3.2 Dataset Specification Methods

Datasets may be defined by the user at two levels - from within the program or at run-time, or both.

The methods for dataset definition are:

1. Specify the device (and file, if applicable) as a dataset within the program at the time of writing.
2. Issue a call from within the program inviting the user to specify devices (and files) at run-time via the Command String Interpreter.
3. Give a dataset a name within the program which is then associated with an actual device (and file) at run-time by means of the ASSIGN command.
4. Give the dataset a name and specify an actual device (and file). The latter is effective for any run for which no ASSIGN command is issued for the dataset; however, an ASSIGN command overrides the dataset specification in the program. It also overrides any values obtained through the Command String Interpreter.

The DOS/BATCH data specification facilities provide great flexibility and complete device independence and interchangeability.

2.3.2.1 Within-program Data Specification

Dataset specification from within the program is effected by control blocks which are known as link blocks and filename blocks.

Link Blocks. For input and output to be device-independent, input or output requests in the program must not directly refer to a device and/or file. Instead, for each dataset which is to be accessed by his program the user sets up in the program a control block known as a link block, which must contain either the name (and number, if applicable) of the device to or from which the dataset is to be transferred, or a name chosen by the user for the dataset, or both. This information may be written into the program or set by the program itself before the dataset is first accessed. All input and output requests in the program then specify not the dataset concerned but the address of the link block which has been set up for that dataset. Therefore if the dataset is changed, only the contents of the link block need be altered or overridden.

Filename Blocks. If the dataset associated with an input or output request is a file, a device name and a filename must be specified. The filename of the file specified as a dataset is held in another control block in the program, known as a filename block.

The filename block is not needed if the dataset is not a file; however, if device independence is required, a filename block must be included in anticipation of a possible later change to a file-supporting device.

2.3.2.2 The Command String Interpreter (CSI)

DOS/BATCH provides a method of defining datasets at run-time, without regard to specific input and output devices. This method is implemented by means of the Command String Interpreter, a Monitor routine that parses and validates dataset specifications and passes them to the calling routine.

The system program that calls the CSI should display the character #. This is an invitation to the user to enter a standard "command string".

A standard command string consists basically of one or more output dataset specifications followed by zero or more input dataset specifications, each consisting of a device name alone (for media which do not support files) or a device name and a filename, as described earlier. The point where output datasets end and input datasets begin is identified by a left angle bracket (<); otherwise datasets are separated by commas. The CSI checks the syntax of the datasets in the command string.

If any check is not satisfied, the CSI rejects the command string; the program should display another #, perhaps preceded by an explanatory message. If the checks are satisfied, the CSI passes a portion of the command string to the program for insertion in its link and filename blocks.

NOTE

Further information or instructions may be given to the program by switches. A switch consists of a character string whose meaning is known to the program, preceded by a slash (/), which identifies it to the Command String Interpreter as a switch. Switches, when specified, must follow the dataset specifier. See Appendix J for a list of switches and their uses.

2.3.2.3 The ASSIGN Command

A program written for specific input and output devices may subsequently be required to run using different devices. It would be inefficient to use the CSI, which requires input from the command source for every run, just to ensure against such a possibility.

DOS/BATCH therefore allows the user to override at run-time the input and output datasets specified in the link blocks, by means of the run-time command ASSIGN. The format of this command is:

```
AS[SIGN] dataset specification,logical name
```

where "dataset specification" is the device name of the device (and, if applicable, the filename of the file) that constitutes the input or output dataset; and "logical name" is the name that identifies the dataset in the link block in the program. For example:

```
AS DT1:NEWDAT,JFM
```

This command assigns the file NEWDAT on DECtape unit 1 to the dataset named JFM in the program's link block.

Similarly, the command:

```
AS MT0:BINOUT,1
```

assigns the file BINOUT on the magnetic tape unit numbered 0 to channel 1 of a FORTRAN program.

The ASSIGN command can also be used to override a dataset specified via the CSI if, for example, a mistake has been made in the command string.

If the input or output dataset is identified by a logical name in the link block, no details of the actual device or file need be included in the program. However, it is recommended that default values be supplied.

NOTE

> If no "logical name" for the dataset has been included in the link block, run-time device assignment by means of the ASSIGN command cannot be accomplished.

2.4 TYPES OF TRANSFER

DOS/BATCH supports the following different input and output requirements:

1. Sequential transfer, to process records one at a time in the order in which they are stored.
2. Random access, to process records (or physical blocks) in random order.
3. Bulk transfer, to transfer large quantities of data without regard to record or file boundaries.

2.4.1 Sequential Transfer

This is the normal type of transfer and the one for which the DOS/BATCH Monitor provides the most support. The user simply issues a request to read or write a record (in the MACRO Assembly Language the relevant program requests are .READ and .WRITE) and the Monitor transfers the next record between the device and the program's buffer (whose size is device-independent) via a buffer within the Monitor.

2.4.1.1 Transfer Modes

In sequential processing the Monitor performs transfers in one of several different ways, as specified by the user. The options available depend on whether the data is binary or ASCII.

1. Formatting. Data can be transferred without formatting, or can be formatted in one of the following ways:
   a. Binary data can be transferred in predetermined numbers of bytes.
   b. ASCII data can be transferred until either a predetermined byte count is reached or a terminator is encountered.
2. Parity. Either of two modes is available for ASCII transfers:
   a. No parity generation or checking, or
   b. Generation of even parity on output and checking on input.

2.4.2 Random Access

The user can access records or physical blocks within a file held on disk or DECtape in random order. Disk and DECtape are the only devices in which the location of a file and its constituent blocks is held in a directory - see Section 2-3.3.4.1. The relevant program requests in the MACRO Assembly Language are .RECRD, which transfers a record of a size specified by the user, and .BLOCK, which transfers one physical block, the size of which is device-dependent and not subject to user control.

Since speed is normally important in random access applications, only one transfer is performed, between the device and the Monitor's data buffer. The user may process the data in that buffer or transfer it to or from his own buffer.

NOTE

Random access is supported for contiguous files only, since they are the only kind of file in which the address of a block or record in a file can be calculated from the start address of the file held in the directory.

2.4.3 Bulk Transfer

As in the case of reading in overlays, large quantities of data can be transferred to or from memory quickly, without examining it or performing any formatting. The MACRO Assembly Language program request for this operation is .TRAN.

Care should be taken with this method of transfer, since it bypasses the file protection provisions described in Section 2-3.4.2.2.

2.5 DESIGN OF INPUT AND OUTPUT ROUTINES

2.5.1 Modularity

Since the management of the transfer of data between the processor and peripheral devices is a major function of an operating system, the total size of the routines dedicated to this task is necessarily large. If they were all held simultaneously in memory, the amount of space that they would occupy would seriously degrade the system. Consequently, the input and output routines should consist of independent modules which can be swapped into memory when required.

In the design of the DOS/BATCH input and output routines, two important facts have been recognized:

1. A particular input/output device may be used infrequently, and most devices are not used at all by any one program.
2. Device independence can best be assured if all input and output requests of one kind use the same routine right up to the point where the actual device becomes involved.

Accordingly routines controlling input and output functions have been carefully separated into two groups of modules as follows:

1. Routines that control common input and output functions whose performance is not concerned with the actual device used.

   a. The .READ/.WRITE processor
   b. The .BLOCK processor,
   c. The .RECRD processor,
   d. The .TRAN processor, and
   e. The special function processor.

2. Routines called device drivers that control physical devices.

2.5.2 Reentrancy

One of the stated objectives of the design of input and output handling was that the program should be able to continue, while an input or output request is being serviced. This offers no problem until another input or output request is reached - which is quite likely to happen before the servicing of the previous request has been completed. This problem has been overcome under DOS/BATCH because the common input and output routines are reentrant; that is, the processing of subsequent input or output requests can start before the previous one has been completed.

PART 2

CHAPTER 3

DATA STORAGE

The term "data storage", as used in this chapter, is defined as the process by which information is placed on a medium in machine-retrievable form for subsequent use on the same or another system.

This chapter describes the facilities, additional to the basic input and output facilities described in Chapter 2-2, which the DOS/BATCH Monitor provides for data storage.

3.1 MEDIA

The first requirement of any storage medium is reliability. Other desirable characteristics are:

1. High capacity,
2. High speeds of access and transfer,
3. Portability, and
4. Low cost.

No one medium can combine all these four characteristics to the highest degree, since some of them are mutually incompatible.

DOS/BATCH supports four magnetic media: disk (both fixed and interchangeable), DECtape, standard magnetic tape and cassettes. Paper tape input and output devices may also be used to store data, though no special support is provided.

The relative strengths of each medium are shown in the following table, in which the devices are ranked from 1 (strongest) to 5 in respect of each characteristic:

	Disk	DECtape	Magtape	Cassette	Papertape
Access Speed	1	2	3	4	5
Transfer rate	1	3	2	4	5
Capacity	2	3	1	4	5

The relative portability of each medium depends on the amount of data involved. Each medium except fixed disk is portable, and is likely to be most convenient for transporting data which cannot be contained on a smaller medium.

Cost comparisons must take into account both the medium itself and the devices needed.

The exact access and transfer speeds and capacities of actual devices are given in Appendix H.

3.2 OBJECTIVES

The Monitor's objectives in providing support for data storage are threefold:

1. To make the best use of the strengths of each medium.
2. To simplify the use of the storage media such that the user should be able to handle his own data easily, and be unaware of and unaffected by other users.
3. To extend the concept of device independence to include interchangeability of data between devices and media (as far as possible).

3.3 MAGNETIC MEDIA

Data storage under DOS/BATCH is supported primarily by magnetic media, namely disk, DECtape, magtape, and cassette tape. The main advantages of magnetic media over non-magnetic media are the much higher transfer speeds and the fact that an equivalent amount of data can be stored in far less space. Consequently, magnetic media are also known as bulk storage media.

3.3.1 Differences between Media

Because standard magnetic tape (including cassette tape) is not as flexible a medium as disk or DECtape, not all of the services provided by the Monitor for the latter media can be implemented on the former. Moreover, those services that are provided on all media are in fact implemented quite differently on disk and DECtape on the one hand, and on standard magnetic tape and cassettes on the other. However, the user interface has been so designed that as far as possible, the user may treat all media alike.

3.3.1.1 Hardware Constraints

On disk and DECtape the recording head can be positioned exactly over any spot required. To take advantage of this ability, data is divided into blocks, and each block of data is placed in a predetermined position whose address is recorded. This arrangement has the following main benefits:

1. Random access to any specified block is made possible,
2. Individual blocks can be overwritten without fear of adjacent blocks being corrupted, and
3. On disk, more than one file can be open simultaneously, and blocks can be accessed from each in turn.

On standard magnetic tape and cassettes, however, such precision is not possible, and no addressing mechanism can be implemented. Instead, records are separated by variable lengths of blank tape. This "inter-record gap" is the only means by which the end of one record and the beginning of the next can be recognized. Switching from reading (required to locate a record) to writing in the middle of a file, is likely to corrupt the inter-record gap, and is therefore not recommended.

3.3.1.2 Monitor Provisions

Because individual records on magnetic tape and cassette cannot be altered, it is not possible to maintain on the medium a directory which is updated every time a file is created, deleted or otherwise altered. It is possible to record the file-name and other information in a header label which precedes the file. When an attempt is made to access a file, a search of the tape must be made to ascertain whether the file exists, whether it belongs to the user, and/or whether he is permitted to access it. A request for a "directory" listing necessitates a similar search.

On disk and DECtape however, all this information and the address at which the file is held, can be held centrally on the medium in a directory, which can be updated as the information changes. Directories are described in Section 2-3.6.1.

NOTE

> For those procedures which are applicable to all devices, a standard interface is presented to the user. For example, if he requests a "directory" listing of a magnetic tape (although it has no directory), a search of all the files on it are listed in the same format as the listing of a disk's directory.

Services provided on all magnetic media are described in Section 2-3.4, and services available on disk and DECtape only are described in Section 2-3.5.

3.4 SERVICES PROVIDED ON ALL MAGNETIC MEDIA

As described in Section 2-2.3.1.2, sets of data are held on magnetic media as files identified by user-chosen names.

3.4.1 User Separation

In order that bulk storage media be fully utilized, it is important not only that each user be able to store several sets of data on one medium, but also that more than one user should be able to use the same medium for data storage. For this to be practicable, each user should be able to choose filenames freely, without risk of confusion if he duplicates names chosen by another user; and each user's files should be protected from other users.

3.4.1.1 User Identification Codes (UIC)

Every user must have an assigned User Identification Code (UIC) by which to identify himself when he logs onto the system. The Monitor, unless otherwise directed, automatically associates every file that the user creates with his UIC. (On magnetic tape the UIC is recorded in the file header label; on disk each UIC has its own directory, into which the names of all files created by the user are entered - see Section 2-3.6.1.) Therefore, if more than one user uses the same filename, each file of the same name is uniquely identified by its UIC. When a user attempts to access an existing file, the Monitor searches for a file of the name specified and having the same UIC as that under which the user is logged in (unless otherwise specified). Any other files of the same name are ignored. Similarly, if a user requests a list of existing files to be output, only those having his UIC are listed (again, unless otherwise specified).

Therefore, the user need not concern himself with other users' files - in fact, he need not be aware of their existence. On the other hand, a user can access a file belonging to another user by specifying that user's UIC with the filename, subject to the restrictions mentioned in the next section.

A User Identification Code consists of two octal numbers, each in the range 1 to 376, separated by a comma and enclosed in square brackets. For example:

```
[130,27]
```

This format is fully explained in Part 3. However it is mentioned here because of the way the first number can be used in connection with File Protection Codes, which are described in the next section.

NOTE

1. A small number of UICs have special roles. In particular UIC [1,1] is associated with system files, including system programs (which are described in Section 2-4.6.2).

2. On DECtape partial user separation is provided, as described in Section 2-3.6.1.1.

3.4.1.2 File Protection

The system of User Identification Codes makes it possible for a user to operate as if his own files and the system's files were the only ones on the system. However, when a user needs to access another user's files, he can specify the file by adding that user's UIC to the filename.

This raises questions of file security and privacy. Some users may not wish their files to be accessed by others. Some may wish only certain of their files to be so accessible. It may be desirable that some files be accessible to other users for reading or running but not for writing or deletion. For that matter, a user might wish to prevent himself from inadvertently overwriting or deleting one of his own files.

DOS/BATCH provides file security and privacy by means of File Protection Codes. Each file's record in the user's directory (or, on magnetic tape, in the file's header label) includes a binary Protection Code, in which each group of bits set prohibits a certain class of user from certain kinds of access.

Four kinds of access are identified, namely running, reading, writing and deletion.

The following classes of user are identified:

1. The owner.
2. The user group. This is defined as all users in whose User Identification Code the first number is the same as in the UIC of the owner.
3. All other users.

For the second and third classes of user specified above, it is possible to prohibit, by means of the File Protection Codes, any of the following combinations of access:

a. Deletion,
b. Deletion and writing,
c. Deletion, writing and reading,
d. All kinds of access.

The owner can prohibit himself from the second combination above, that is, deletion and writing.

NOTE

These facilities do not provide absolute protection for a user's files, since no password mechanism exists and it is therefore possible to log in under another user's UIC. Also the .TRAN Program Request (see Section 2-2.4.3) ignores all control information including file protection codes. However protection is provided against inadvertent access to user's files by normal access methods.

3.5 SERVICES PROVIDED ON DISK AND DECTAPE

3.5.1 Directories

Directories are the means whereby a file can be accessed without a search of the medium being necessary. They are held on the same medium as the files which they reference, and are of two kinds, namely User File Directories and Master File Directories.

3.5.1.1 User File Directories (UFD)

A User File Directory is a list of a user's files. The directory entry for each file contains the file's filename, location and protection code - in other words, any information which is about the file but not part of it.

Before a user can create a file of his own, his User Identification Code must have been associated with a UFD. The association of a UIC with a UFD is a function of the system program PIP, which is described in Part 12. For both disk and DECtape, the user simply enters his UIC onto the medium by supplying the appropriate command string to PIP. However the association of UICs with UFDs is implemented differently on disk and on DECtape.

1. On disk, a new UFD is created for each user. Each UFD is given a name which consists of the UIC under which the user logged in. All files subsequently created under that UIC are entered in the UFD of the same name unless otherwise specified. Complete user separation is thus achieved.

2. On DECtape user separation can be readily achieved physically, by each user having his own DECtape. The need for the Monitor to separate users on one DECtape is not sufficient to outweigh the overheads involved. Therefore each DECtape has only one UFD, which is shared by all users who enter their UICs onto the medium. All files on the DECtape are treated as belonging equally to all such users, but are protected in the usual way from other users.

When a user specifies an existing file, unless he also specifies a UIC different from one under which he logged in, the Monitor will in most cases search only the UFD with which his UIC is associated. (If the file specified is to be run, the system UFD [1,1] is also searched.)

3.5.1.2 Master File Directories (MFD)

The locations of the UFDs on the medium and the names of the UICs with which they are associated are held in a Master File Directory (MFD), which the Monitor uses to implement the procedures described above. Thus on each medium there is a two-level directory structure, as illustrated in Figure 2-1.

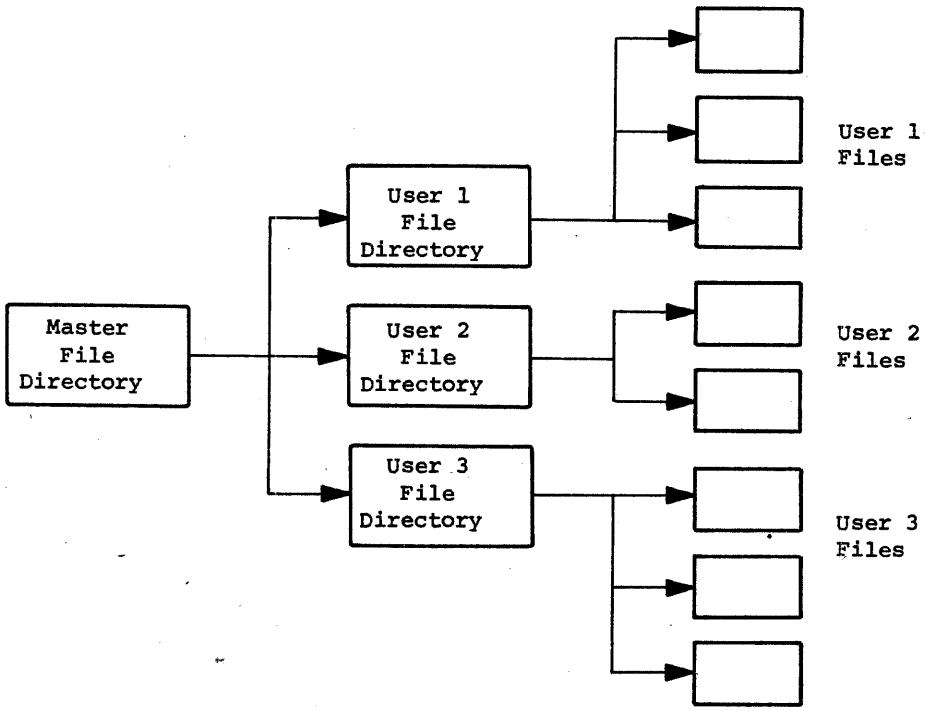

Figure 2-1
Disk Directory Structure

3.5.2 Types of File

To support the different kinds of input and output processing, two types of files are provided on disk and DECtape, namely contiguous files and linked files.

3.5.2.1 Contiguous Files

The most straightforward kind of file is one on which the data is recorded on consecutive blocks until the file is complete. Since the blocks are physically adjacent, such a file is known as a contiguous file.

Contiguous files are ideally suited to random access (through .RECRD or .BLOCK - see Section 2-2.4.2). In random processing the address of an item of data to which access is required has no necessary relationship to that of the last item accessed. Therefore, in order that any item can be reached in minimal time, irrespective of the current position of the medium, it is essential that the absolute address of the item can be readily determined from a relative value supplied by the user. This requirement is satisfied by the use of an area of physically contiguous blocks on the medium. The calculation of the actual block required is simple provided that the start block of the area is known, and provided that the items being accessed are of the same length. Restricting operations on the file to a compact area on the medium also affords optimal access, particularly on moving-head disks because of the reduced need for head-positioning and on DECtape because of its inherent linear nature. This advantage is reduced if other files on the same medium are being accessed simultaneously.

To create a contiguous file on any medium, a contiguous area large enough for the file must be available. The user requests the space when creating the file and must therefore know, or guess, the size of the file before it has been written. Once the contiguous file area has been established it cannot be extended, since there is no guarantee that the requisite adjacent blocks are available; nor can such files be joined together unless they happen to be adjacent to each other. A user request for either operation is rejected by the Monitor. However, the user of contiguous files is not limited to random access. If he opens the file properly, he can process his data sequentially by .READ or .WRITE. The Monitor ensures that such operations are effected only within the preset file-bounds.

Contiguous files are convenient for bulk transfer (more than one block) by means of .TRAN. The Monitor itself transfers contiguous files in this way.

3.5.2.2 Linked Files

Normal data transfer is sequential, implemented by means of the program requests .READ and .WRITE. For this kind of access the user does not need to know the size of his files in advance, but he does need the ability to extend or concatenate files. Therefore the DOS/BATCH Monitor supports another kind of file on disk and DECtape, called a linked file. When reading a linked file, the Monitor does not assume that the next logical block in the file is the next physical block on the medium (although it may be). Instead each block contains a word, known as the link-word, in which the address of the next block is held. This word is not seen by the user.

A 4-block interval (known as the "interleave factor") is left between blocks on DECtape in order to provide ample space for the tape to stop and restart before the next block required reaches the recording head. For hardware efficiency when writing a linked file on DECtape the Monitor normally spaces the blocks of the file a constant number of blocks apart; on disk consecutive blocks are used. In either case, if the next intended block is already in use the Monitor simply finds another one, and records its address in the link-word of the previous block in the usual way. Therefore, there is no restriction on the file's size other than the total capacity of the medium.

A file can be extended, or two files can be concatenated, without any movement of data. The Monitor simply places the address of the beginning of the extension, or file to be appended, in the link-word of the last block of the existing (first) file.

Linked files are not designed for random access since the only means by which this can be effected is by a perhaps lengthy search along the links of the file for the required block. The random-access requests .RECRD and .BLOCK are therefore illegal for linked files, as noted in Section 2-2.4.2.

3.5.2.3 Mixing Contiguous and Linked Files

Linked and contiguous files may share the same medium; however, some conflict in block availability eventually occurs, even though the medium may not actually be filled. To delay this as long as possible, linked files are set up at the front of the medium or low-address end, and contiguous files are given space starting from the high-address end. If the conflict does occur, as evidenced by a failure to find sufficient space for a new contiguous file, the user must either delete some of his other files or transcribe them in order to utilize more fully any smaller disconnected gaps between used blocks.

3.5.3 Verifying Directories with VERIFY

Although reasonable precautions are taken, efficiency of normal operation precludes a system of absolute file security. Therefore for reasons outside the control of the Monitor (e.g., erroneous use of .TRAN by an incompletely tested user program, or a system failure which precludes the writing of critical information held in memory) corruption of the file structure can occur. To protect the user against a proliferation of errors, the system program VERIFY is provided.

Normally VERIFY confirms that the file structure is self-consistent. If discrepancies are found, however, VERIFY reports the following:

1. Lost blocks (blocks shown to be used but not allocated to one file).
2. Blocks allocated to more than one file.
3. Actual and recorded lengths of files.

Facilities are provided for restoring the self-consistency of the file structure. See Part 14 for more detailed information on VERIFY.

3.6 SERVICES PROVIDED ON MAGNETIC TAPE

Although magtape is not considered a file structured device, certain structure and label processing features have been implemented to enable creation and retrieval of multiple files on a magtape.

Files on magtape consist of a 7-word label record which contains identification information including user protection via UIC's which is followed by sequential data records bounded by an end-of-file record.

Special system requests are available which permit the user to position a magtape on line. These requests include:

1. Skip Records - forward space over the requested number of records.
2. Rewind - rewind the magtape to the beginning of tape.
3. Set Buffer Size - allows the user to specify a record size different from the system default.

3.7 SERVICES PROVIDED ON CASSETTE

Cassette support is in general implemented in the same way as for magnetic tape; that is, the filename is held in a header label. However, there is no user separation on cassettes.

3.7.1 Multi-volume Files

On cassette an option is available whereby if the end of tape is reached while a file is being written, the file may be continued on one or more additional cassettes. Multi-volume files may be read only in the order in which they are written.

3.8 NON-MAGNETIC MEDIA

3.8.1 Paper Tape

The only non-magnetic medium for which both input and output devices are available under DOS/BATCH, and which can be used for the storage in retrievable form of computer-produced data, is paper tape. This inexpensive medium is suitable for the storage of small sets of infrequently used data, or as a backup medium if the installation has no magnetic tape devices. However, compared to magnetic media, paper tape is bulky and slow, and repositioning of the recording head is not possible. In addition, it requires more physical handling and two separate devices for input and output. No facilities other than those described in Chapter 2-2 are provided by DOS/BATCH for data storage on paper tape.

3.8.2 Punched Cards

DOS/BATCH supports input from, but not output to, punched cards. This medium is therefore suitable for holding data prepared off-line, particularly batch streams (see Section 2-6.3.2.4), since cards can be inserted, removed and rearranged as required.

3.9 SUMMARY OF FACILITIES ON EACH MEDIUM

The facilities supported on each medium are summarized in the following table:

	Disk	DECtape	Magtape	Cassette	Papertape	Punched Cards
Named files	x	x	x	x		
User separation	x	partial	x			
Sequential access	x	x	x	x	x	x
Bulk transfer	x	x	x	x	x	x
Random access	x	x				
Extendible files	x	x				

3.10 MEDIUM INTERCHANGEABILITY

As described in Section 2-2.3.1.1, the DOS/BATCH input and output routines are device-independent. This is also true of the file-processing routines. A corollary of this is that sets of data, once output, are also medium-interchangeable - that is, they may be transferred freely from one medium to another. (This interchangeability is naturally subject to the provision that the destination medium has sufficient space and can support the type of file being transferred.)

3.10.1 Manipulating Data with PIP

PIP (Peripheral Interchange Program) is a system program (see Part 12) which performs the following operations:

1. Copy a dataset to any medium.
2. Concatenate two or more datasets held on any media into a single dataset on any medium.
3. On disk and DECtape:
   a. Associate a User Identification Code with a User File Directory,
   b. List User File Directories,
   c. Delete a file or files,
   d. Rename a file,
   e. Change a file's protection code.
4. On magnetic tape and cassette, print directory-type listings of a user's files.

3.10.2 Conversion from EBCDIC with EBASCI

Data held in the Extended Binary Coded Decimal Interchange Code (EBCDIC), which is not normally supported by DOS, may be converted into ASCII by means of the system program EBASCI. EBASCI accepts data in EBCDIC format held on magnetic tape and converts it into ASCII. Output may be to any suitable device.

Magnetic tape with ANSI standard labels or with no labels are accepted. The inclusion of carriage return and line feed characters after each logical record is at the option of the user. See Part 16.

PART 2

CHAPTER 4

MEMORY MANAGEMENT

One of the chief functions of an operating system is the management of memory. It is not practical to attach to the processor enough memory to hold all the code, including the operating system itself, that may be needed to run any job. For this reason, a disk is an essential part of the DOS/BATCH Operating System. During operation, the disk can hold code, particularly parts of the operating system itself, until that code is required. The code can then be quickly transferred into memory. The part of a disk that is used in this way is known as "secondary storage".

The overall objective of memory management is to make maximum memory space available to the user's program and data, by designing and managing the other occupants of memory so that the minimum possible space in memory need be set aside for them.

The operating system should manage with maximum efficiency both the available memory and the transfer of code to and from secondary storage.

4.1 MEMORY OCCUPANTS

The following are the potential occupants of memory for which space must be provided:

1. Running programs, originating from:
   a. The user
   b. The system.
2. Incoming and outgoing data.
3. The stack (see Section 2-4.5.1).
4. The Monitor itself.
5. Drivers for input and output devices. These are part of the Monitor, but their memory occupancy is organized differently from that of other parts.

Under the DOS/BATCH Monitor items 3 to 5 of the above list typically occupy less than 4K words of memory; the remainder is available to user (or system) programs and data.

4.2 MEMORY MANAGEMENT CRITERIA

The methods used by the Monitor to save space vary in detail for each occupant of memory, but the following are the main criteria which apply throughout:

1. Frequently used code should be held in main memory, and less frequently used code on secondary storage.
2. Code held on secondary storage should not be allocated space in memory until it is needed, and the space should be freed again as soon as the code occupying it has fulfilled its function.
3. The user should be able, both when the system is generated and in his program, to specify code that is to be held in memory.
4. Swapping between secondary storage and memory, though necessary, should be kept to a minimum; maximum use should be made of available memory. In particular, it is important to avoid having small pockets of unused memory which are wasted.

4.3 OVERALL STRATEGY

Of the potential occupants of memory, three must be held there for the duration of a program's run - namely, the program, the stack, and the Resident Monitor (see Section 2-4.4.2). The area required for the Resident Monitor remains fixed in size once it has been loaded; the size of the user's program may change during execution. These two components are therefore placed at opposite ends of memory. The Resident Monitor, since it must be loaded first, and because it requires access to hardware vectors, is placed at the bottom (that is, starting at address 0), and the user's program (or system program - see Section 2-4.6.2) at the top. This leaves a single area which is available to the stack and other potential occupants of memory. Since the presence of these other occupants is controlled by the Monitor in accordance with the dynamic requirements of the running program, this central area is referred to in this manual as "dynamic memory".

These three main areas of memory are illustrated in Figure 2-2. The rest of this chapter describes the way in which these areas are managed by the Monitor.

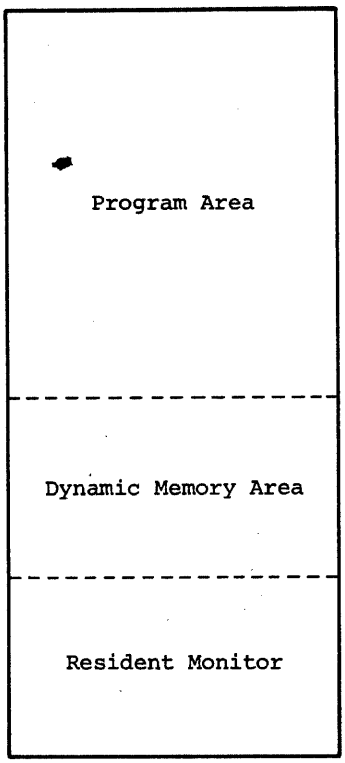

Figure 2-2

Main Areas of Memory

4.4 THE MONITOR AREA

Of all the potential occupants of memory, the Monitor itself is one of the largest and most complex. The amount of memory that the Monitor occupies and the manner in which it organizes its occupancy are critical to the overall performance of the system.

4.4.1 Monitor Modularity

To achieve economy of memory usage the Monitor has been constructed as a set of separate modules, each of which has a specific function. Most of these modules are not held permanently in memory, but are held on secondary storage and overlaid into memory when required. This concept is the key to the design, and the success, of the DOS/BATCH Operating System. (This is reflected in the first part of its name, "Disk-based Operating System".)

This modularity has two great advantages from the production point of view. Groups of modules can be written and tested independently, making for greater reliability, and future extensions to any one module or the addition of new modules need not affect the others. The user benefits in both cases.

The following subsections describe how the criteria listed in Section 2-4.2 have been applied to the Monitor modules.

4.4.2. The Resident Monitor

That part of the Monitor which remains permanently in memory is known as the Resident Monitor.

4.4.2.1 The Minimum Resident Monitor

As a minimum the Resident Monitor consists of the following:

1. Modules that control other modules or the system in general,
2. Monitor tables,
3. The system device driver, which must be resident to handle the swapping of other modules from the system device,
4. The keyboard listener, which handles interrupts from the keyboard, which may occur at any time,
5. The .READ/.WRITE processor (See Section 2-2.5.1),
6. Interrupt vectors,
7. Swap buffers, which hold disk-resident Monitor modules (with some minor exceptions) when they are overlaid into memory. These two are:
   a. The main swap buffer,
   b. The keyboard swap buffer.
8. The clock handler (if a clock is present).

All the above satisfy the criterion of frequency of use. The layout of the minimum Resident Monitor is illustrated in Figure 2-3.

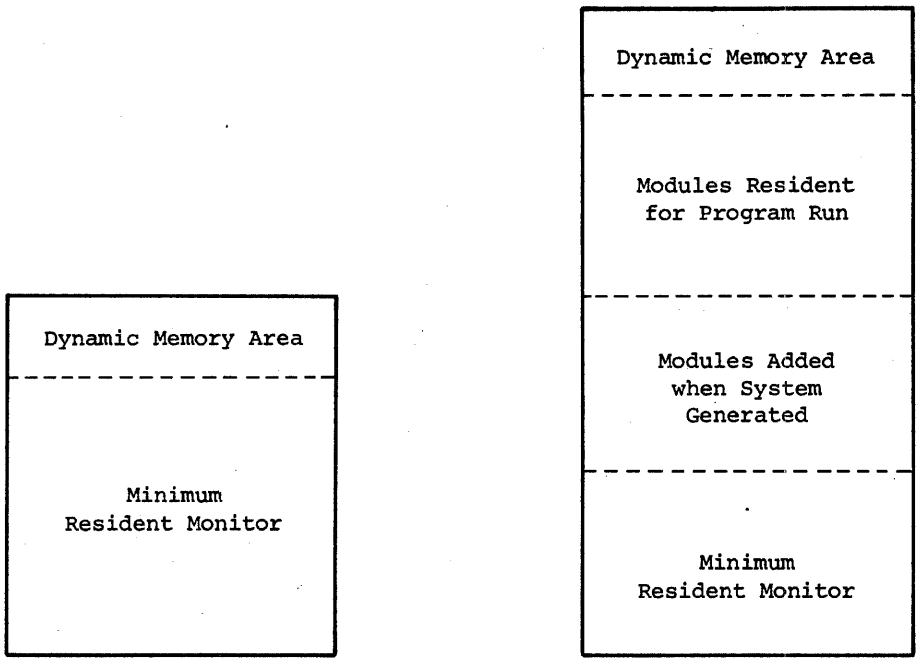

Figure 2-3

The Resident Monitor

4.4.2.2 Extending the Resident Monitor

The only advantage of having Monitor modules non-resident is memory economy; at an installation with a larger than average amount of memory it may be preferable to have more Monitor modules in memory and so save overlaying time. Even on an installation with an average amount of memory, if a particular module is required very frequently by most users it might be worthwhile to include that module in the Resident Monitor. Therefore provision is made for the inclusion of additional modules when the system is generated (see Section 2-8.3.1). This is not, however, permitted for device drivers (see Section 2-4.4.4.2).

4.4.2.3 Making a Module Resident at Run Time

Alternatively, if an individual user requires a particular module to be resident throughout the running of a particular program, he may indicate his requirement as a global reference in the program. If the module specified is not already permanently resident, it is added by the Program Loader to the end of the Monitor, and is not removed until the program has finished its run.

Programs which include such a requirement may be run on any system, because these Monitor modules are not added until run-time. However, this facility should not be used unnecessarily, since the size of the dynamic memory area is correspondingly reduced. The location of modules resident for a program run is illustrated in Figure 2-3.

4.4.3 Non-resident Monitor Modules

The secret of the success of the Monitor in providing so many services and yet occupying only a minimal portion of memory is that most Monitor modules are not resident in memory but are held on secondary storage until needed. With the exception of device drivers, whose management is discussed in Section 2-4.4.4.2, non-resident modules, when they are needed, are placed in one of the two swap buffers within the Resident Monitors. Modules in this category are the common input and output routines described in Section 2-2.5.1, the keyboard service routines, and the utility routines described in Chapter 2-5.

4.4.3.1 The System Device

To prevent the actual process of transfer from seriously degrading the system, non-resident modules should be held on a medium that allows both fast access to the module required and fast transfer of the module into memory. Disk has both the fastest access speed and the highest transfer rates. The non-resident Monitor modules, together with copies of the resident modules, are normally held on this medium. The Monitor may be held on any type of disk (see Appendix H), but in multi-disk programs the Monitor must be held on the unit numbered 0.

The device selected to hold the Monitor is known as the "system device" or the "system disk". The system device also stores program data in the ordinary way (see Chapter 2-5). In fact if it is the only disk on the installation, it will be heavily used for this purpose.

The system device is handled differently from other devices in the following ways:

1. Its driver is permanently resident in memory. This is necessary to access non-resident Monitor modules, but it also has the advantage that access times are reduced, not only for the system device but for every other device of the same type, since an initial transfer of the driver itself is obviated.

2. System device transfers take priority over others, since they may be required to control others.

3. A file held on the system device may be specified by its filename alone because the default value for a device name (see Section 2-2.3.1.3) is the system device. Not only does this save time - it also provides device independence because dataset specifications need not be changed if the system device is changed (this is particularly helpful for batch streams - see Section 2-6.3.2).

4.4.4 Input and Output Routines and Device Drivers

The functions of input and output routines and device drivers are described in Chapter 2-2. This section describes how the occupancy of memory by drivers and other input and output routines is managed. Although drivers are Monitor modules they are discussed here separately because their memory occupancy is handled differently from that of other modules.

4.4.4.1 Common Input and Output Routines

The input and output routines which effect the different kinds of data transfer are listed in Section 2-2.5.1 Each routine is used for all transfers of the appropriate kind irrespective of the physical device involved - that is, the routines are common to all devices for which they are used. Of these common routines, the .READ/.WRITE processor, which is the largest and most commonly used, is held permanently in memory as part of the Resident Monitor. The other input and output routines are handled like other non-resident Monitor modules; that is, they are held on the system device and overlaid into the main swap buffer within the Resident Monitor when required (see Section 2-4.4.3).

NOTE

Input/output routines are likely to be heavily used. The reentrancy mentioned in Section 2-2.5.2 carries the advantage that processing of a second input and output request can start before the previous one has been completed, without duplication of routines being necessary. This assists considerably in memory economy.

4.4.4.2 Device Drivers

The system device driver is held as part of the Resident Monitor, occupying, typically, about 15Ø decimal words. Device drivers other than the system device driver are held on secondary storage until required, when they are transferred to an area set up in dynamic memory as described in Section 2-4.5.2. Since the drivers consist only of code related to that particular device, and do not contain code duplicated in other drivers, their size is kept to a minimum; hence the areas reserved for them do not need to be very large.

NOTE

Unlike other Monitor modules, device drivers may not be incorporated in the Resident Monitor when the system is generated, as described in Section 2-4.4.2.2, nor can they be made part of the Resident Monitor for the duration of a program in the way described in Section 2-4.4.2.3. However, this is not a practical restriction since a driver may be called into memory from within the program for any length of time required (see Section 2-4.5.2).

4.5 THE DYNAMIC MEMORY AREA

The area between the user's program and the Resident Monitor is referred to as "dynamic memory"

Space is allocated in the dynamic memory area for the following:

1. The stack,
2. Incoming and outgoing data (I/O buffers),
3. Device drivers,
4. Temporary Monitor tables.

Of the above occupants of the dynamic memory area one, the stack, is resident for the duration of a program's run but is variable in size. The others though individually fixed in size, are resident only temporarily; hence their collective size is variable. The method used to make the best use of dynamic memory, and avoid the creation of useless "pockets", is illustrated in Figure 2-4.

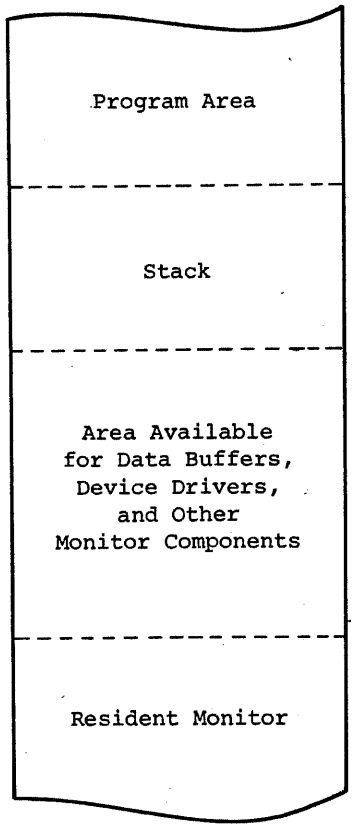

Figure 2-4
The Dynamic Memory Area

The stack is placed at the top of dynamic memory, next to the user's program, where it can expand downwards. Allocation of space for the other occupants begins at the bottom of dynamic memory, just above the Resident Monitor, and proceeds upwards, so that there is only one area of unused memory.

NOTE

To save space in the Monitor's internal tables the total area available for data buffers and device drivers is limited to half the memory area. However, since it is unusual for a program to require Monitor dynamic storage larger than the program itself, this is unlikely to be a practical restriction.

4.5.1 The Stack

Next to the user program is an area of memory known as the "stack", for the temporary storage of data - in particular, data which is to be transferred between two routines within the program, or between the program and a Monitor routine.

The stack functionally has characteristics similar to those of a stack of physical objects. As many items of data as required may be added to or removed from the stack, but only at the top.

The advantage of this arrangement is that it is necessary for the Monitor to hold only one address for the stack - that of the item currently at the top. This address is held in a register known as the Stack Pointer (SP). The address of an item lower in the stack may be calculated by counting the number of intervening items.

NOTE

> The terms "top" and "base" when applied to the stack are used as they would be in an analogous stack of physical objects; that is, the "base" is the stationary end and the "top" is the end to and from which items may be moved. The terms "top" and "base" in this context are not connected with the relative positions of the two ends of the stack in memory. In fact, the base of the stack is adjacent to the program area, and the stack expands downwards - that is, each item added has a lower address than the previous one.

4.5.2 The Data Buffer and Device Driver Area

Space in the dynamic memory area for data buffers and device drivers is allocated and released in the following manner:

1. When the program indicates that a series of transfers is about to take place, the Monitor allocates space from dynamic memory for the driver, a data buffer, and any necessary control blocks. The amount of space required is known to the Monitor.
2. When the program notifies the Monitor that no more transfers are required, the sections allocated are released and so become available for use on another occasion.

It should be noted that buffers and drivers are not reshuffled when the lower parts of dynamic memory are released, because of the overheads that would be involved, and because all references to them would then have to be indirect. The user can minimize the creation of unused pockets by issuing requests for routines which cause memory allocation in the opposite order to that in which the space will be released.

4.5.3 Temporary Monitor Tables

Certain temporary Monitor tables which control data transfer operations are also allocated space in the Dynamic Memory Area when required.

4.6 THE PROGRAM AREA

The ultimate objective of the management of the Monitor-occupied area and the dynamic memory area, is that as much of the computer's memory as possible should be available to user programs.

4.6.1 User Programs

The user program is loaded by default into the top of memory. The Monitor places no restriction on the way the program uses this area, but naturally there is a limit to the size of the program. This limit is the total amount of memory less the amount occupied by the Resident Monitor, and a reasonable allowance for dynamic memory.

4.6.1.1 Overlaid Programs

The limit to the size of the user program that can be held in memory at one time can be overcome by the technique of overlaying. In principle this technique is the same as that applied by the Monitor to its own overlaid modules as described in Section 2-4.4.1. To make use of overlays it is necessary to observe a set of rules (described in Part 9) when referencing parts of the program that might be overlaid. The overlays are then created by the system program LINK in accordance with the user's instructions (see Section 2-7.2.3).

4.6.2 System Programs

Chapters 2-2 and 2-3 described the services provided to running programs by the DOS/BATCH Monitor. These services are built into the Monitor-in fact, the Monitor may be regarded as a collection of run-time services.

Non-run-time services such as data copying between devices and program development are provided through system programs. To perform specific frequently required services, system programs are provided in ready-to-use form which are available to all users.

The provision of such services by means of system programs rather than as an integral part of the Monitor has the following advantages:

1. Memory management. System programs can occupy the program area in memory and be handled like user programs.
2. Selectivity. Only those system programs actually required need be included in the system. Others may be added as required, without modification of the Monitor.
3. Flexibility in development. System programs may be improved, and new ones introduced, without disturbance to the system as a whole.

System programs operate in the same way as user programs. That is, they occupy the top of memory and interact with the Monitor as user programs do. They are often modular in structure and may make use of the technique of overlaying.

4.6.3 The Transient Monitor

One of the basic functions of the Monitor is the loading of programs (both user and system programs). As suggested in Section 2-5.1, loading a program involves a number of ancillary activities - for example, the stack must be moved. Some of these activities must, and others may be performed before the actual loading of the program as a whole takes place. Similarly after a program is unloaded it is necessary to restore memory to a state in which it is ready for another program.

There is no need for the routines which carry out these activities to use the swap buffers one after another when the whole of the program area is available for their use.

The routines which occupy the program area when no program is loaded are known collectively as the Transient Monitor. The Transient Monitor performs the following functions when a program (user or system originated) is to be loaded:

1. Restores memory after program unloading.
2. Accepts operator instructions.
3. Implements those operator commands, which, by definition, operate when there is no program in memory.
4. On receipt of an appropriate command, carries out preparations for the loading of the next program.

PART 2

CHAPTER 5

UTILITY ROUTINES

Although the actual processing of data, as distinct from its input and output is the domain of the program, there are some procedures which are common to many programs; it is desirable that the program should be relieved of handling the details of such procedures.

5.1 PROGRAM LOADING AND UNLOADING

The loading routines perform the following functions:

1. Check that the dataset specified as a program or overlay exists and is accessible to the user.
2. Load into memory any Monitor modules required to be resident (see Section 2-4.4.2).
3. Prepare the stack and the rest of memory to receive the program.
4. Load the program.
5. If required, run the program.

When a program is removed from memory, Monitor routines release any dynamic memory (see Section 2-4.5.2) which was acquired on behalf of the program, and restore any necessary system control data.

5.2 CHARACTER CONVERSION

A number of conversion services are provided by the Conversion Utilities Package.

5.2.1 ASCII/Binary Conversion

Numeric data is normally entered into the computer either for arithmetic processing or for storage, or both.

Arithmetic operations are performed in binary (which is also the most economical form for storage) but input and output are more convenient in either octal or decimal numbers, in the form of ASCII characters.

Routines are therefore provided to perform the following conversions:

1. Binary to decimal ASCII.
2. Binary to octal ASCII.
3. Decimal ASCII to binary.
4. Octal ASCII to binary.

5.2.2 Radix-50 Packed Character Storage

A large portion of non-arithmetic data consists of symbolic names, such as filenames. ASCII code is stored one character per byte or two characters per word. However, because symbolic names are formed from a restricted group of 39 characters (letters, digits, space, period, dollar sign) it is possible to store such names three characters to a word. Conversion in each direction is performed by the Radix-50 Pack and Unpack routines by means of a single algebraic formula.

5.3 SYSTEM INFORMATION

The Monitor holds a large amount of information about the state of the system. The General Utilities Package consists of routines whereby the user may either access this information or supply additional information to the Monitor.

Routines are provided to set any of the following to values specified by the user:

1. The stack base address,
2. Certain system vectors.

Routines are provided to ascertain the current value of the following:

1. The stack base address.
2. Certain system vectors.
3. The total amount of memory available.
4. The amount of memory occupied by the Monitor.
5. The amount of memory occupied by the Monitor and its buffers.
6. The name of the system device (see Section 2-4.4.3.1).
7. The User Identification Code under which the program is running (see Section 2-3.4.2.1).
8. The date (internal or ASCII representation).
9. The time (internal or ASCII representation).

5.4 FILE MANAGEMENT

Routines are provided to perform the following file management functions:

1. Find a file and supply information about it.
2. Change the name or protection code of a file.
3. Append one file to another.
4. Delete a file.
5. Allocate a contiguous file.

PART 2

CHAPTER 6

USER CONTROL

The previous chapters have described DOS/BATCH Monitor input and output, data storage and other services to the program, and memory management. An implicit requirement of all management activities is that they be conducted in accordance with the user's needs and wishes. This chapter describes user-Monitor communication.

6.1 COMMUNICATION BETWEEN USER AND MONITOR

The user can issue instructions to the Monitor at either or both of two levels, by the following means:

1. Instructions from within a program written in FORTRAN or the MACRO Assembly Language.

   a. FORTRAN statements. A FORTRAN statement that specifies a channel number, for example

      ```
      WRITE (5,1)
      ```

      has the effect of sending output to the line printer (which is assigned to channel 5 unless overruled by an ASSIGN command - see Section 2-2.3.2.3). For further details see Part 7.

   b. MACRO Program Requests. The MACRO Assembly Language contains more than forty Program Requests which may be issued from within the program to make use of Monitor services in any of the areas of activity described in the previous four chapters. For further details see Part 6.

2. Run-time Commands. A comprehensive command language permits the user to communicate his requirements at run-time by means of commands issued either interactively from the console keyboard, or in the form of a batch stream from any input device.

The Monitor can communicate with the user to inform him of the effect of the commands he has issued or of program events. In general, expected events are signalled by standard responses which invite the next input, whereas unexpected events are reported by means of error messages. These responses are described in Section 2-6.4.

6.2 MACRO PROGRAM REQUESTS

A program written in the MACRO Assembly Language can issue a Program Request to call on the following Monitor services.

1. Input and output services. These consist of the data transfer operations and the suspension of processing pending completion of transfer.
2. File management services such as providing information about existing files, opening, closing, deleting, concatenating or renaming files, or allocating contiguous files.
3. Memory management services such as reporting the size of memory, providing the addresses of the current occupants of memory, or changing the stack base address or program load address.
4. Processing services such as setting vectors or the program restart address, loading a program or overlay, converting binary or Radix-50 values to ASCII characters (or vice versa), or supplying the date or time.

MACRO Program Requests consist of up to five letters or digits preceded by a period - for example .OPEN, .CSI2. They are described in full in Part 3.

6.3 RUN-TIME COMMANDS

The user can call on a large range of Monitor services at run time, by means of a comprehensive command language.

Run-time commands can be issued either singly from the keyboard or in batches from any suitable input device. In the former case the process is interactive, in that the user awaits the Monitor's response to one command before issuing the next; in the latter case the sequence of commands is predetermined, and when the Monitor has completed the action initiated by one command it reads and implements the next.

The commands are easy-to-remember plain English words and to save time and avoid errors, only the first two letters of each command need be typed. (Indeed the Monitor does not examine any subsequent letters.) In the case of the RUN command, either RU or even R alone is accepted.

Many of the Monitor actions which the user may wish to initiate will be the same irrespective of the command source, and therefore, a large part of the Command Language may be used from either input source. However, since batch operation uses a superset of the keyboard language, two modes of operation are described separately.

6.3.1 Interactive Operation

Since programs under development will often behave in unexpected ways, interactive operation allows the user to adapt his actions to deal with what does happen. This may also be true when he is using standard programs, for example PIP, to perform tasks for which no established procedures exist.

DOS/BATCH, controlled interactively from the keyboard, is ideally suited to such applications. The command language can be used to respond constructively to any event, expected or unexpected. Commands are included to load, suspend, continue, stop, restart, alter and save user programs, run system programs, assign and release datasets, and control the user's relationship with the system. These commands are fully described in Part 3.

6.3.2 Batch Operation

The chief disadvantage of interactive operation is the low speed at which a user can type his input. This limitation can be overcome by means of the batch facilities which are provided by the DOS/BATCH Monitor (and after which the Monitor is partly named).

6.3.2.1 Applications

Batch operation offers outstanding advantages for two distinct applications.

One application is for a large number of users - for example students for whom there is not sufficient time to run programs in turn from the keyboard. Each user prepares his commands as a batch stream off-line on punched cards or paper tape. Each stream is then run by means of a single command from the keyboard in a fraction of the time required for on-line typing. Batch operation greatly increases machine utilization.

The other application is for repeated use of the same command sequence. The commands may be made up into a batch stream stored in a file like any other data, and used and reused as often as required. Initial input may be off-line or at the keyboard by means of the editor. In either case the saving, in terms of machine and operator time and the avoidance of errors, is such that batch operation may be regarded as indispensable for regular production runs.

6.3.2.2 Implementation

Batch operation is implemented in two complementary ways.

First, many of the same commands that are issued from the keyboard may be stored on any medium and issued subsequently from an input device. The keyboard commands unsuitable for batch use are discussed later.

Second, special batch commands are provided, which:

1. Manage batch operations. These commands are described in Section 2-6.3.2.5.
2. Perform standard functions, such as compiling, linking and running programs and copying, listing and deleting datasets, which would otherwise require one or more system programs to be explicitly run and supplied with command strings. Commands in this class are known as "concise commands". They are described in Section 2-6.3.2.6.

6.3.2.3 Input Media

A batch of commands may be set up on any medium for which an input device is available. Which medium is most suitable depends on the purpose for which batch operation is being used. The commands may be prepared off-line on punched cards or paper tape; alternatively they may be entered from the keyboard into a file by means of the editor and the file may be stored on any suitable magnetic medium; or, thirdly, both methods can be combined; that is, the batch stream can be prepared off-line and copied into a file by means of PIP.

6.3.2.4 Batch Streams

A batch stream consists of a string of commands held on any suitable medium. Such a batch stream may be specified as a dataset.

Most of the keyboard commands can be included in a batch stream. However, the usefulness of certain commands depends on their being used interactively. In particular, several keyboard commands deal with error conditions; these commands range from CONTINUE, which orders the program to resume after an interruption, to ODT, which initiates the ODT on-line debugging program. Such commands are inappropriate for a batch stream and, if included, are either ignored or reported as errors depending on the command.

The KILL command, which removes a program from memory, is largely redundant in a batch stream because commands that might follow a KILL are implemented to automatically kill a completed program as well as perform their own functions. This principle of providing the means for economizing on commands is further extended by the provision of "concise" commands, which are described in Section 2-6.3.2.6.

6.3.2.5 Batch Mode

The BATCH command BATCH, instructs the single keyboard Monitor to read a specified dataset and obey the commands contained in it. For example:

```
$BATCH JOBRUN.BAT[7,4],LP:
```

specifies that the file JOBRUN.BAT, owned by user 7,4, is to be run as a batch stream, and that the commands in the batch stream, as well as any error messages, should be printed at the line printer.

When the Monitor obeys a BATCH command it is said to enter batch mode, and user control is exerted from the batch stream. However, a batch stream still must recognize keyboard commands. An example is the use of the CONTINUE command to resume processing after a peripheral has been made available by the operator. The operator must be able to interrupt execution of the batch stream at any time.

Other commands are provided specifically for the management of batch processing. The command CHANGE, when included in the batch stream, enables a subsidiary batch stream to be read and run. This ability to run two levels of batch stream is important. It can be used, for example, by creating a master batch stream consisting entirely of CHANGE commands each of which runs another batch stream, the whole run being initiated by a single BATCH command from the keyboard.

6.3.2.6 Concise Commands

Concise commands combine the effects of more than one standard command, or of a command to run a system program and a command string supplied to that program. They thus form a higher level command language.

For example, the following concise command:

```
$CPY DT1:FILEA TO PP:
```

replaces the following sequence:

```
$RUN PIP
#PP:<DT1:FILEA
```

Either of the above causes the file FILEA held on DECtape unit 1 to be output on paper tape.

Similarly the following concise command:

```
$EX[ECUTE]
```

replaces the sequence:

```
$RUN FORTRAN
#MYPROG,LP:<BI:/GO
```

Either of the above causes the FORTRAN program MYPROG held in the batch stream (BI:) to be compiled, linked and run, and a listing of the source program to be printed on the line printer.

Concise commands are described in Part 4.

6.4 MONITOR REPORTING

To exercise control effectively the user must be well informed about the progress of his job.

6.4.1 Interactive Reporting

6.4.1.1 Normal Responses

The simplest way in which the Monitor reports progress during interactive operation is by means of "ready" signals. A "ready" signal is output at the Keyboard device to inform the user that processing has ceased, and to invite further input. In order to provide further guidance to the user, different "ready" signals are used to indicate different situations, as follows:

- $ A command is required. Either the previous command has been unsuccessfully implemented or an error has occurred and corrective action is required. In the latter case the $ is preceded by an error message.
- . Processing has been interrupted by the user from the keyboard and that a command either to resume or to kill the program is expected. (This signal is normally output only when a program is loaded.)

The system program invites a command string to be processed by the Command String Interpreter. (It is recommended that user programs should follow the same convention.)

* The program expects command input to the program other than a command string. For example, the system program EDIT uses it to invite editing commands.

6.4.1.2 Error Messages

If an event occurs that the user did not intend, he wants to know what has gone wrong. Therefore the Monitor provides two kinds of error messages:

1. Command error messages are output when a command cannot be implemented, and indicate the reason. Examples are:

Message	Meaning
INV CMD!	Invalid command
NO FILE	The file cannot be found, or cannot be loaded.

2. Coded messages identify error conditions arising during the execution of a command or program.

   Each message begins with a letter to indicate the class of error.

   - A Action required by the operator.
   - F Fatal error. The program has been terminated.
   - I Informative message only.
   - S System program error.
   - W Warning to the operator.

   The initial letter is followed by a three-digit code which identifies the error; see Appendix K. Additional information may also be included in the message.

Also, system programs issue messages, many of them self-explanatory.

6.4.2 Batch Reporting

Errors arising from commands in a batch stream are printed at the keyboard in the normal way. The user may specify in the BATCH command that a log, consisting of each line of the BATCH stream and any error messages generated, be output to any device. In this way progress can be monitored and any error messages can be seen in context.

PART 2

CHAPTER 7

PROGRAM DEVELOPMENT

The services provided by the DOS/BATCH Monitor itself and by the system program PIP provide ample support for the running of established programs. However, the development of a program is much more complex than its subsequent running, and the support required from the operating system during this process is correspondingly much greater.

In most cases a user program is not running when the services needed for program development are required. Such services are therefore provided through system programs.

7.1 INPUT OF SOURCE PROGRAMS

Source code written in FORTRAN or the MACRO Assembly Language may be put into the computer in either of two ways, as described in the following subsections.

7.1.1 Direct Input through EDIT

Source code may be inserted directly into a file from the keyboard, and then checked and corrected by means of the system program EDIT.

EDIT is a general purpose text editor with a straightforward powerful editing language. Facilities are provided for inserting, deleting or altering one or more characters or lines of text. The required point in the text may be located by means of line and/or character counts or string searches. Sections of one or more lines may be moved or duplicated. An editing command or a string of commands may be stored and executed either a predetermined number of times, or until the end of the file is reached. See Part 8 for more details.

NOTE

EDIT is suitable not only for programs but for any text, including documents. The text of this manual, for example, was initially punched onto paper tape, and then revised by means of EDIT.

7.1.2 Offline Preparation

Source code may be prepared off-line on punched cards or paper tape, and copied onto a magnetic storage device by means of the system program PIP (see Part 12). It may then be edited, as described above.

7.2 COMPILATION, ASSEMBLY AND LINKING

Under DOS/BATCH a source program is converted into runnable machine code in two distinct stages:

1. Individual segments of source program are converted into object code. This process is known as compilation for high-level languages such as FORTRAN, and assembly for assembly languages such as MACRO. The resulting modules are called object modules.

2. These individual modules, which may have been compiled or assembled at different times, are then combined to form a new module through a process known as linking. The resulting module is ready to be loaded and run, and is called a load module.

The two-stage conversion has the following advantages:

1. Segments of programs can be compiled or assembled independently. (Small segments are easier to comprehend and to maintain.)
2. Standard routines can be held in libraries as object modules and linked into the program after compilation or assembly.
3. Program segments can be written in different languages.

NOTE

Filename extensions (see Section 2-2.3.1.4) identify the source program, object module, and load module states of a program. The Monitor supplies the following standard extensions to distinguish the different states of a program.

FTN	FORTRAN source file
MAC	MACRO source file
OBJ	Object module
LDA	Load module

Thus, for example, when assembling the module COMPAR, the user can refer to both the input and the output files as COMPAR, while the assembler processes them as COMPAR.MAC and COMPAR.OBJ, respectively.

7.2.1 FORTRAN IV Compilation

The DOS/BATCH FORTRAN IV Compiler fulfills all the requirements laid down by the American National Standards Institute (ANSI) for FORTRAN in 1966. Programs written on other systems in accordance with this standard can therefore be run under DOS/BATCH. See Part 7.

7.2.1.1 Extensions

DOS/BATCH FORTRAN offers certain features not specified in the ANSI standard, in particular the following:

1. Octal data type.
2. The following mixed mode arithmetic expressions:

    a. Real with integer,
    b. Double precision with integer,
    c. Complex with integer.

3. Random access data transfer (define file).
4. In-core ASCII/binary conversion (encode/decode)

7.2.1.2 Optimization

At the user's option, code can be optimized to take advantage of the processor on which the program is to run and to make the best use of the extended integer arithmetic and floating point hardware if they are available. If the user does not choose to optimize his code, he may elect to have additional run-time error-checking performed.

Optimization can achieve significant improvements in running speeds for both floating-point and non-floating-point computation - in some cases run time may be halved.

7.2.2 MACRO Assembly

For those cases in which it is desirable to work closely with the instruction set of the PDP-11 and where a compiler is not to be used, DOS/BATCH provides a powerful assembler, called MACRO. As its name suggests, MACRO provides extensive macro facilities (see Part 6). A comprehensive set of system-defined macros is available, and in addition the user may define his own macros. At the user's option, segments

of code may have either absolute or relative addresses; in the latter case relocation is undertaken by LINK. Other important features are a full set of conditional assembly directives and a cross-reference capability.

7.2.3 Linking with LINK

The system program LINK links program segments to each other when they have been successfully compiled or assembled, and at the same time, if required, incorporates routines held in a library.

Link performs the following functions:

1. Relocates each object module and assigns absolute addresses.
2. Links the modules by correlating global symbols defined in one module and referenced in other modules.
3. Implements an overlay structure defined by the user.
4. Produces a load map which displays the absolute addresses assigned.
5. Creates a load module which can be loaded and run.
6. Writes the load module as a dataset rather than putting it in memory. This allows it to be used more than once.
7. Provides a cross-reference listing for globals.

For more specific information, see Part 9.

7.2.3.1 Linking Overlaid Programs

In order to decide on an overlay structure it is necessary to know which segments are required in memory simultaneously. This is difficult if the program has not been written. Under DOS/BATCH the overlay structure need not be declared until the program is linked. A particular advantage of this is that if the overlay structure first chosen turns out to be unsatisfactory, a different structure may be tried without changing the program.

At the user's option overlays may be loaded explicitly by the program (manual loading) or automatically when called by the program (load on call).

An example of an overlay structure is given in Figure 2-5.

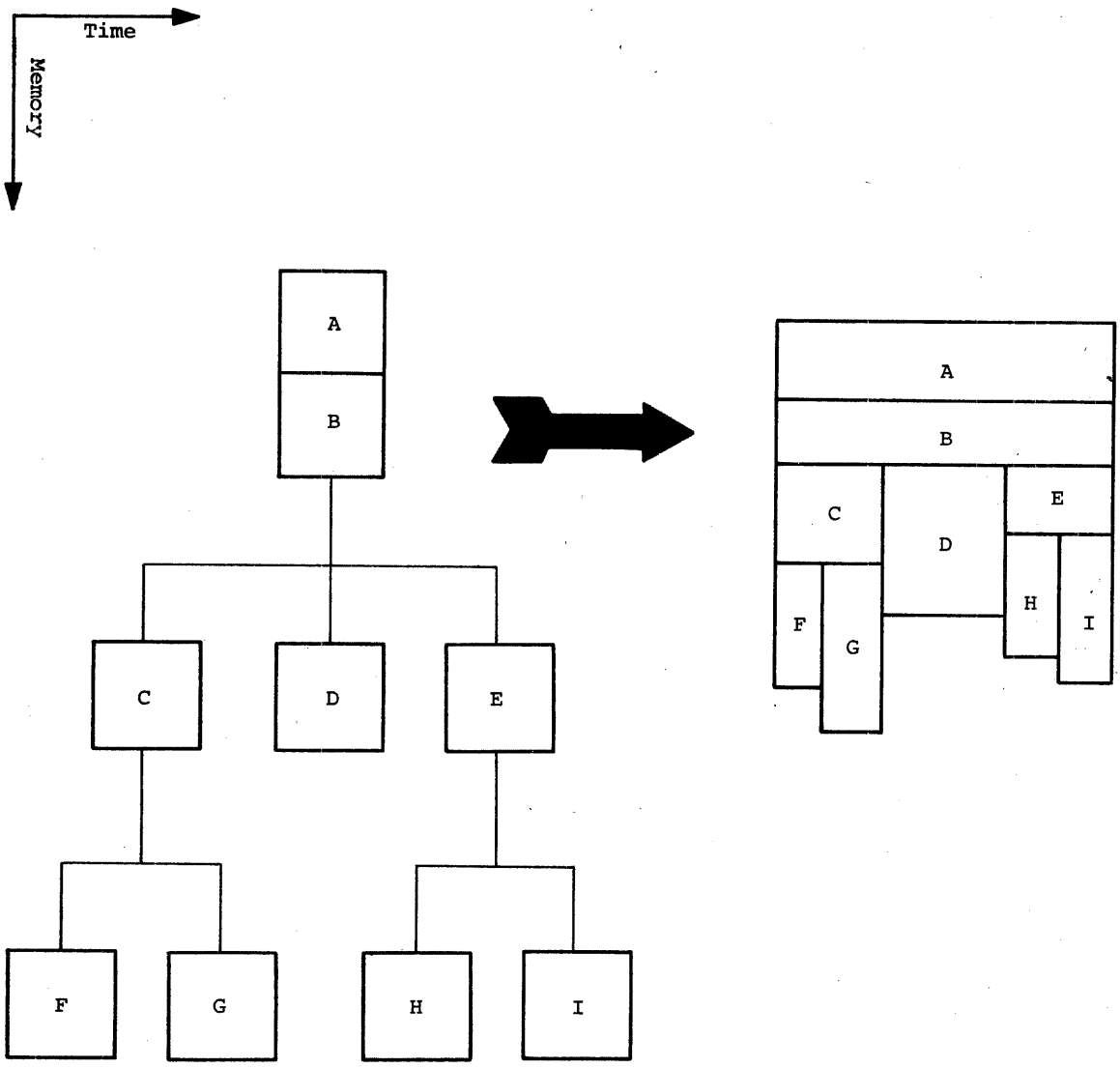

Figure 2-5

Example of an Overlay Structure

7.2.4 Library Building with LIBR

Programs may be assembled or compiled in separate segments as described in the previous section, but it is not always efficient to hold all the resulting object modules in separate files so that each has to be recorded individually in the user's directory and each has to be specified individually to LINK. The system program LIBR can be used to create a single "library" file into which the object modules of the program can be inserted.

A primary example of the technique of library building is the FORTRAN library, which contains commonly required FORTRAN routines.

7.3 DEBUGGING

Several system programs provide services to aid in the debugging of user programs under development.

7.3.1 Editing with EDIT

Any required changes to a source program can be made through the system program EDIT, which is described in detail in Part 8.

7.3.2 Dumping Files with FILDMP

The contents of a file may be examined by means of the system program FILDMP. FILDMP can read the whole file or specified blocks of a file consisting of formatted or unformatted binary data or formatted ASCII data; can create a dump copy in octal bytes or words or ASCII characters; and can, optionally, treat each word as a group of three packed Radix-50 characters, and print the characters represented. More than one representation of the file may be specified. Output can be sent directly to the keyboard device or line printer or be stored in a file. See Part 15 for more details.

7.3.3 Comparing Files with FILCOM

The file comparison program FILCOM can trace changes to a file by providing a list of differences between two specified ASCII files. For more details see Part 14.

7.3.4 The On-line Debugging Program ODT

The system program ODT (On-line Debugging Technique) enables the user to:

1. Print the contents of any location or register.
2. Change the contents of any location or register.
3. Set breakpoints anywhere in the program to interrupt execution and give the user control at the keyboard.
4. Perform single-step execution.
5. Search the program for specific bit patterns.
6. Search the program for instructions that reference a given word.
7. Ascertain offsets for relative addresses.
8. Fill a block of words or bytes with a specified value.

ODT is unique among system programs in that it is loaded concurrently with the user program which is being tested. The two programs run co-operatively and interact with one another.

PART 2

CHAPTER 8

SYSTEM DISTRIBUTION, GENERATION, AND MODIFICATION

One of the advantages of DOS/BATCH is the ease with which it can be installed and set into operation. This chapter describes the methods used for the distribution and building of the system.

8.1 DISTRIBUTION MEDIA

DOS/BATCH is distributed on each of the following media:

1. DECtape
2. Magnetic tape (7- and 9-track)
3. Disk cartridge

8.2 MONITOR FORMAT

The first task when loading the Monitor is to copy the whole Monitor, including the part that will ultimately be resident in memory, onto the system disk.

8.2.1 Core Images

The loading procedure described in Section 2-5.1 is satisfactory for a program that is run infrequently. However, if the whole procedure had to be repeated every time a Monitor module was swapped into memory the consequent system degradation would be intolerable. Therefore the Monitor modules are held on the system device not as load modules but as "core images"; that is, exact replicas of their appearance in memory without any control information. When it is required, a core image can be simply copied straight into memory.

8.2.2 The Monitor Library (MONLIB.CIL)

The Monitor must be a single coherent file and must have direct access to individual modules so that they can be loaded with minimum delay. This individual accessibility is also necessary when the Monitor library is being updated.

The requirements of Monitor unity and individual module accessibility are satisfied by a Core Image Library (CIL). Just as libraries of object modules can be built by means of LIBR so libraries of core images can be built by means of the system program CILUS (Core Image Library Update and Save).

A Core Image Library is a file consisting of a set of core images and a directory which holds the address of each image. The file is named, and can be referred to in the usual way. Individual core images can be accessed by the Resident Monitor when a module is to be overlaid, and by CILUS when changes are required. Since the Monitor must load many non-resident modules (some many times) during a typical run, it must know in advance where to find each. This is most easily accomplished if the modules can be accessed directly; therefore the Monitor library is a contiguous file. An additional advantage is that modules which require more than one block for storage can be loaded in a single transfer.

The Monitor, then, is held on the system device in the form of a Core Image Library known as the Monitor Library, whose filename is usually MONLIB.CIL (.CIL is the standard extension for a Core Image Library).

8.2.3 The File MONLIB.LCL

The Monitor Library is held on each distribution medium in a form appropriate to that medium - for example, on magnetic tape it is held as an ordinary sequential file. It can easily be converted into a contiguous file when it is copied on to the system device.

On DECtape the Monitor Library is held as a linked file. A Core Image Library held as a linked file is known as a LICIL (Linked Core Image Library). The standard extension for a LICIL is .LCL, and so the name of the Monitor Library on DECtape is MONLIB.LCL. In order that a standard interface be presented to the program that transfers the Monitor to the system device, the same name is given to the Monitor Library on all distribution media. Similarly the file is often referred to as LICIL irrespective of the medium on which it is held.

8.3 ESTABLISHING THE SYSTEM

8.3.1 Building the Monitor With SYSLOD

The program SYSLOD transfers the Monitor from the distribution medium to the system device. SYSLOD itself is held on the distribution medium and is loaded manually via the operator's console.

SYSLOD performs the following functions:

1. Clears the disk selected as the system device and initializes it to the DOS/BATCH file structure.

2. Copies the Monitor onto the specified system device, converting it to a contiguous file.

3. Unless specified otherwise, loads the Resident Monitor into memory and transfers control of the system to it.

8.3.2 Building System Programs

The system programs are usually distributed as object modules so that they can be built to take advantage of the amount of memory available on the machine on which they will be running. Once the Monitor is running, the system programs can be linked under the Monitor control by running batch streams held on the distribution medium.

8.3.3 Preserving the System with ROLLIN or PIP

The whole contents of the system device, including the Monitor system programs and user files, can be copied onto magnetic tape and subsequently back onto the system device by means of the program ROLLIN. This provides backup in case of a system crash. Alternatively PIP may be used. See Part 16 for more detailed information on ROLLIN.

8.4 MONITOR MODIFICATION

It may on occasion be necessary to modify the Monitor. For example, if a new peripheral device is added to the system, a driver for that device must be included in the Monitor. It is also possible to reconfigure the Monitor, for example, by specifying that various Monitor modules become part of the Resident Monitor.

Since the Monitor is modular such modification is simply performed by the substitution of one module for another, or by the inclusion of an additional module. This is done by the system program CILUS.

8.4.1 CILUS (Core Image Library Update and Save)

The system program CILUS can be used to perform any of the following functions:

1. Build a core image library (as a contiguous or linked file) from specified core images (load modules).

2. Copy a LICIL onto disk, converting it into a contiguous file.

3. By means of a simple editing language, edit a core image library by deleting, inserting, replacing or renaming specified modules.

4. Load and run the first core image in a core library.

5. Determine which core image library should be loaded and run the next time the system is started.

6. List the directory of the core image library.

7. Produce an octal dump of one or more modules.

Full details of the procedures for establishing and modifying the system are given in the System Manager's Guide.