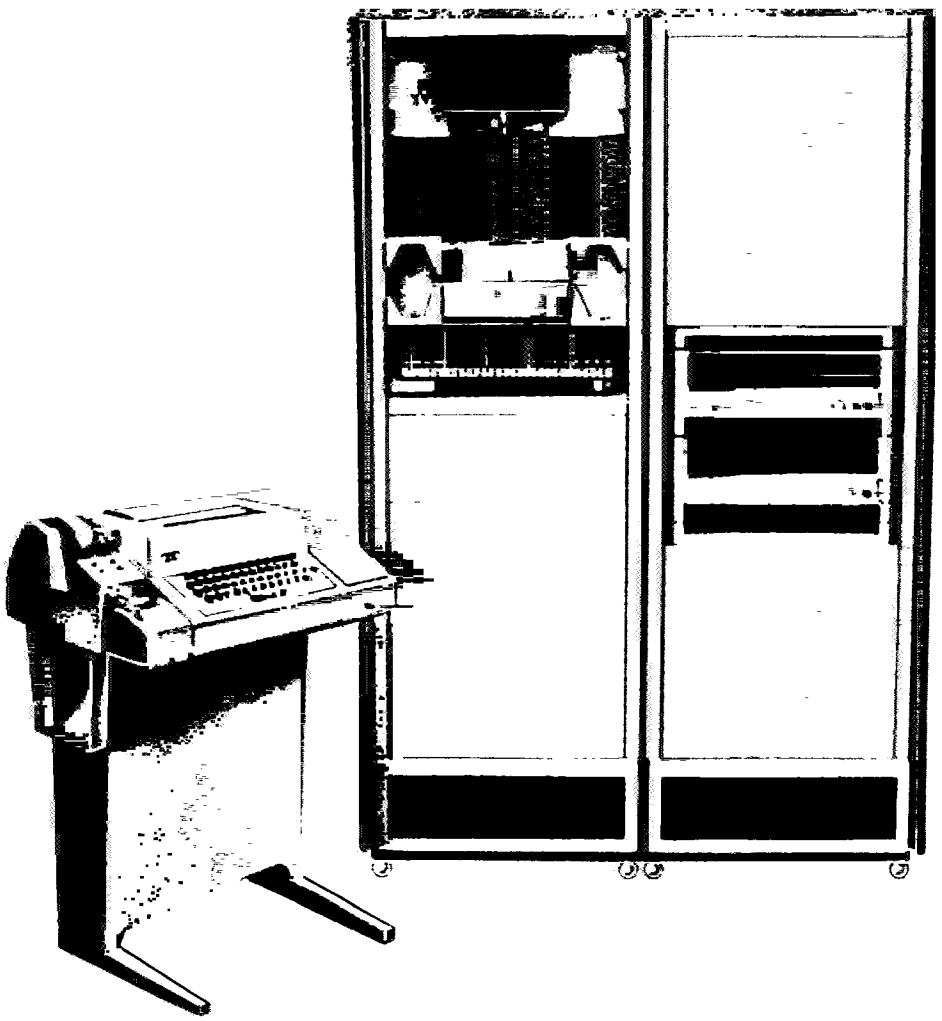
DISC Operating System (DOS) Reference Manual

INTERDATA®



DISC Operating System (DOS) Reference Manual

Publication Number 29-293R01

INFORMATION CONTAINED IN THIS MANUAL IS SUBJECT TO DESIGN CHANGE OR PRODUCT IMPROVEMENT

PRINTED IN U.S A. • JULY 1973

DISC OPERATING SYSTEM REFERENCE MANUAL

TABLE OF CONTENTS

TABLE OF CONTENTS
(Continued)

ILLUSTRATIONS

TABLES

APPENDICES

CHAPTER 1
SYSTEM CONCEPTS

1.1 INTRODUCTION

The objective of an operating system is to provide a method of program management that, by establishing conventional responses to external and internal stimuli, creates an environment in which user programs can function more nearly as pure problem solvers. The Disc Operating System (DOS) creates such an environment. This manual describes the standards and conventions, and the convenience features of DOS that allow the user to program efficiently by concentrating more on the problem at hand and less on the tasks of input/output processing and interrupt handling.

1.2 FEATURES

A significant feature of DOS is that it allows programs to be device independent. That is, I/O calls to DOS do not specify any device dependent information such as physical address and special format characters. Programs written to run under DOS do not have to be modified when hardware is reconfigured or when new hardware is added. DOS fully supports the FORTRAN IV Compiler and the INTERDATA Assembler. It is a modularly organized system containing built-in drivers for Teletype, Disc, High Speed Paper Tape Reader/Punch, Card Reader, Line Printer, Magnetic Tape, and Cassette Tape. User written driver programs for special devices can easily be added to the system. DOS provides the user with convenient and efficient methods of using an INTERDATA disc system, as a program preparation tool, a program library, and an application data storage device. DOS, by means of a conditional generation procedure, may be installed tailored to a user's hardware configuration; i.e., only those routines needed by the user are included in the system.

Features of the Disc Operating System (DOS) are:

Concurrent Input/Output. I/O can be performed in both overlapped (I/O proceed) and non-overlapped (I/O Wait) modes. All interrupts are automatically handled by the DOS executive.

Logical To Physical Device Independence. The user may, with minimum constraints, transfer logical control from one physical device to another without modifying his program. Up to 16 user logical units (files) may be open at any one time. The number of logical units available on a system can be established at System Generation time.

Batch Operation. Operator commands to DOS may be accepted from any input device including catalogued disc files, allowing the user to batch job control statements. (e.g., FORTRAN Compile and GO operation.)

Named Files. The DOS allows the user to have up to 400 named files on a disc pack.

Allocation. DOS allows the user to initially allocate desired space to a disc file and automatically allocates further disc space up to three cylinders to satisfy user requirements. This automatic overflow allocation is invisible to the user.

File Manipulation. The DOS allows files to be named, examined, and manipulated. The user may copy a disc file to another disc file, or may append a disc file to another disc file. In addition, files may be deleted and all files may be listed with appropriate parameters.

File Attributes and Protection. Associated with each disc file is a set of attributes which the user may specify. These attributes include read and write protection plus data and file descriptors. The file protection feature allows a user to save a file and protect against unauthorized access to that file.

Disc Access Methods. The DOS provides three access methods: sequential, random, and direct physical access. Sequential mode is normally used to access records consecutively such as in compilations and assemblies whereas random access mode allows accessing a particular record by specifying a logical record number. Random access is useful in updating master files on disc and in accessing large tables that are kept on disc. Direct physical access allows reference to logical sector numbers within a file, and permits rapid transfer of variable length records which can be any number of sectors.

Variable Physical Record Length. The physical record size on disc may be varied at system generation time to best suit the user's application. Physical record sizes may be 256, 512, 768, or 1,024 bytes. A small physical record size is best suited for random applications, while a large record size increases throughput in sequential operations.

Logical Record I/O. DOS read and write requests to the disc are performed via logical records. DOS performs all blocking and unblocking of physical records based on the specified logical record length of the file and the disc physical record size. In all cases the user simply makes an I/O request for a read or write operation (specifying the logical record number for random files).

Disc I/O via Supervisor Call. The user program performs all disc I/O via a SVC 1 instruction, which passes appropriate parameters to the DOS executive. Specific operations which can be performed are described later but basically the user specifies the operation as Read/Write, Wait/Proceed, Random/Sequential, or as a specific control operation (i.e., Rewind).

Disc I/O Buffering. To insure a minimum of accesses, the user may specify the length of the logical records within a physical record as well as the access method. DOS determines if a logical record is already buffered in core rather than do an unconditional access, and if a Write operation is requested, DOS places the logical record in the DOS buffer and outputs the buffer only when necessary. In a sequential file, a Read operation is not performed prior to sequential Write operations.

Overlays and Program Callable Programs. User programs running under the DOS can issue a SVC to fetch a named overlay into memory or load a program and transfer control to that program. When an overlay is loaded, control is returned to the main program. This capability is useful in FORTRAN programs which are too large for a memory size but, when divided into a main program and one or more subroutines, can fit if the main program is resident and the subroutines can overlay each other. The generation of overlays with the linkages to the main program and library routines is handled by the OS Library Loader.

FORTRAN Programming with Random Accessing. A FORTRAN program may access a specific logical record by calling a disc positioning subroutine (POSITN) prior to executing a Read or Write statement.

Disc Pack Interchange. The DOS allows the interchange of disc packs since all the disc pack control information required by the DOS is resident on the disc pack.

DOS Bootstrap. A Bootstrap tape, Program Number 07-046, is provided for loading DOS from disc. A core image of DOS is kept on Cylinder 0 of each disc pack.

System Generation. The user has the capability of tailoring his system to his own configuration and use, by means of a System Generation assembly, thus assuring a compact DOS environment.

Compatibility. Programs which run under the INTERDATA Basic Operating System (BOSS) execute under DOS with no modifications.

Operator Commands via Supervisor Call. DOS has a feature that allows operator commands to be executed via a program supervisor call. A user program while running may, for example, allocate, open, and close files.

1.3 ELEMENTS OF THE SYSTEM

DOS is a modular system, made up of many elements which contribute to the whole system. The basic DOS modules—the Executive, the Supervisor, the Loader—and the various drivers combine to produce the complete DOS system. Refer to Figure 1-1.

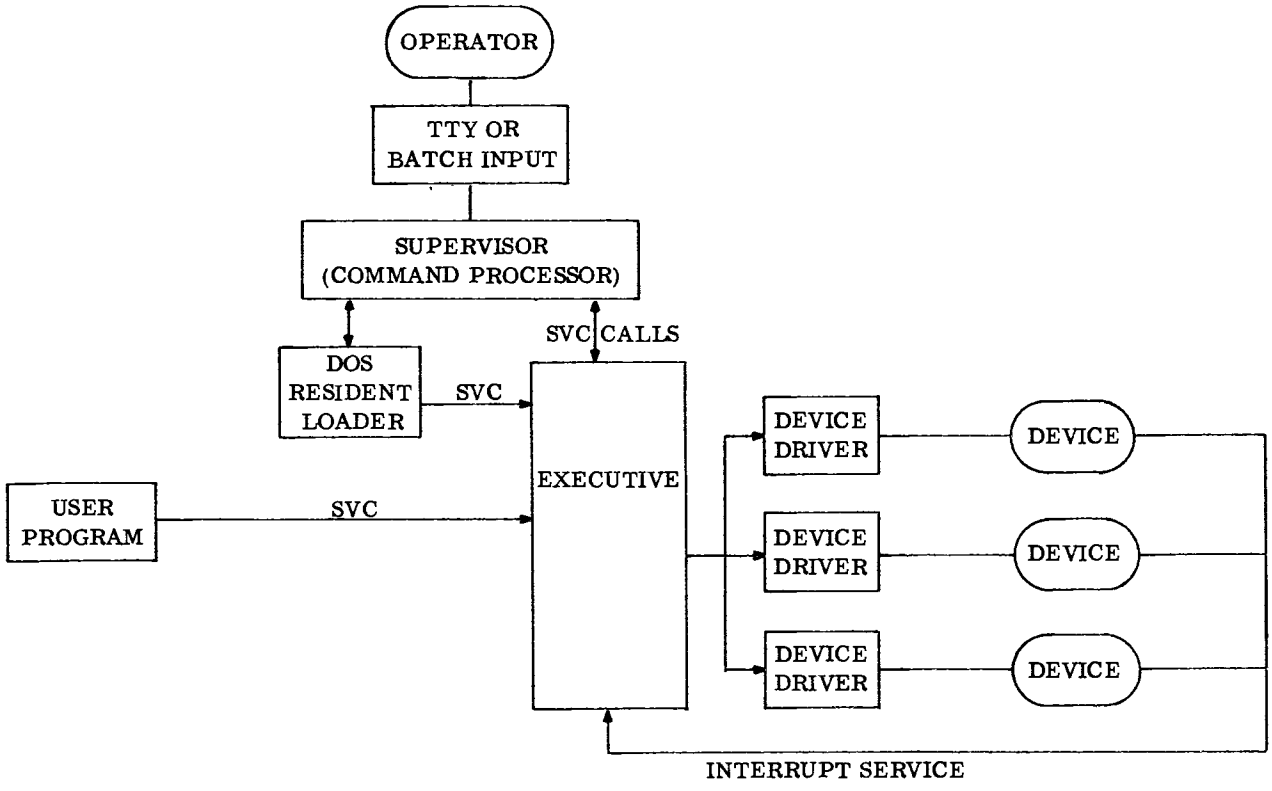


Figure 1-1. DOS System Elements

1.3.1 Executive

The executive handles all internal interrupts; supervisor call, arithmetic fault, illegal instruction, queue termination, and machine malfunction. In its handling of the supervisor call interrupt, it services all requests and initiates all I/O operations. The executive is always entered as the result of an internal interrupt. All executive routines exit to the user program state (PSW Bit 7 set).

1.3.2 Supervisor

This module is responsible for operator message logging, and for operator command processing. The supervisor issues SVC calls to the executive portion of DOS. Operator commands are provided for:

System Initialization	- PACK
Program Execution Controls	- LOAD, CONTINUE, BIAS, START, TRANSFER, RUN, CLOSE, HALT
Device Controls	- BSFM, FRFM, REWIND, WTFM, ASSIGN, LIST UNITS
Operator Debug Aides	- OPEN, REPLACE, PLUS, MINUS
Disc File Management	- ALLOCATE, ATTRIBUTE, DELETE, LIST, ACTIVATE, POSITION
System Utility	- COPY

1.3.3 Loader

The DOS resident loader can be called by the operator or by a running program. It loads programs from the system library or any sequential device.

1.3.4 Device Drivers

These routines, the unique coding sequences that control actual I/O transfers, are activated and terminated by the executive. Drivers are written such that interrupts are disabled for the shortest possible period of time, thus enabling the system to remain responsive to external interrupts even while the executive is running, thus increasing throughput.

1.3.5 User Programs

A user program may consist of a single program, or it may include a main program and a number of subprograms and overlays. User programs may be permanently resident in memory, or they may be loaded from the system library as required. The total number of user programs allowed in memory at any given time is limited only by the amount of memory available, but only one program may be in execution at any given time.

1.3.6 Other Programs

The OS Library Loader accepts its own set of operator commands and runs as a user program under DOS. It loads, links, edits and produces object modules and overlays. The Hex Debug program contains features useful for debugging (i.e., breakpoints) and various utility features such as memory save and memory dump. For details on these programs see the appropriate program descriptions.

1.3.7 DOS Directory

The DOS Directory is a sequential file on Cylinder 1 of each disc pack containing the parameter information of all named files on that pack. It is used by the supervisor for file management operations.

1.3.8 Memory

Figure 1-2 shows a typical memory map for a DOS System. DOS itself occupies the lower memory locations. Within this area are the permanently resident system modules.

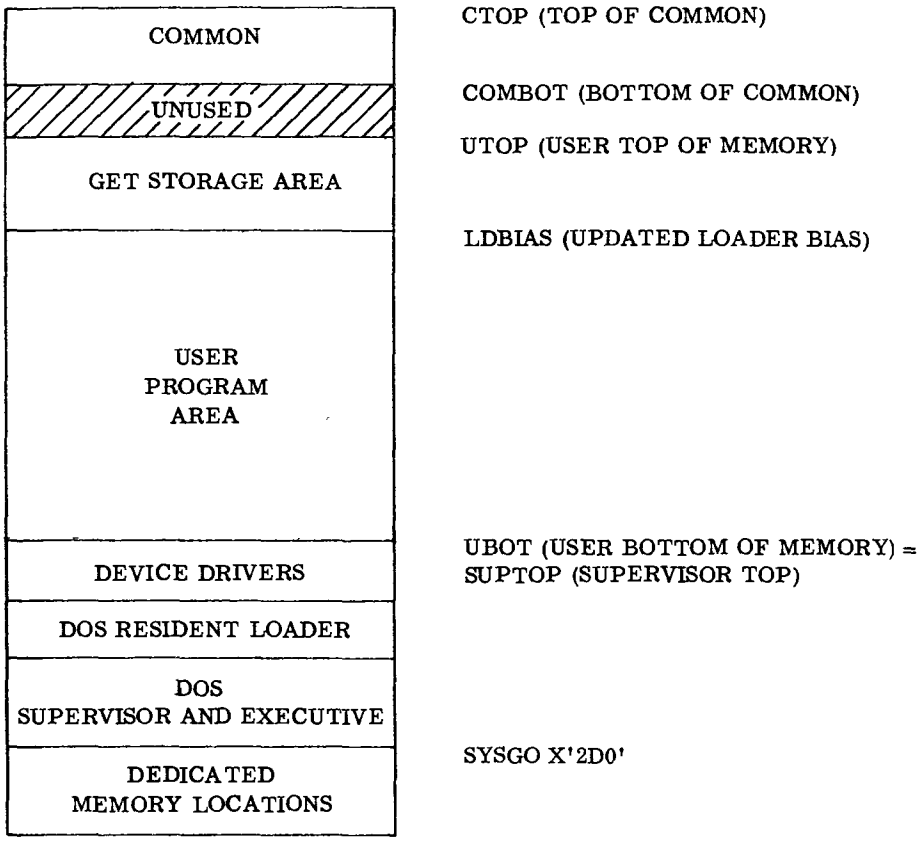


Figure 1-2. DOS Memory Map

1.4 SYSTEM CONFIGURATION

1.4.1 Hardware Configuration

The minimum hardware configuration to support DOS with DISC is:

Model 5, 50, 70, 74 or 80 Processor

16KB Memory

Teletypewriter

Disc Controller with one Disc File (on a Selector Channel)

Refer to Figure 1-3 for a typical DOS configuration.

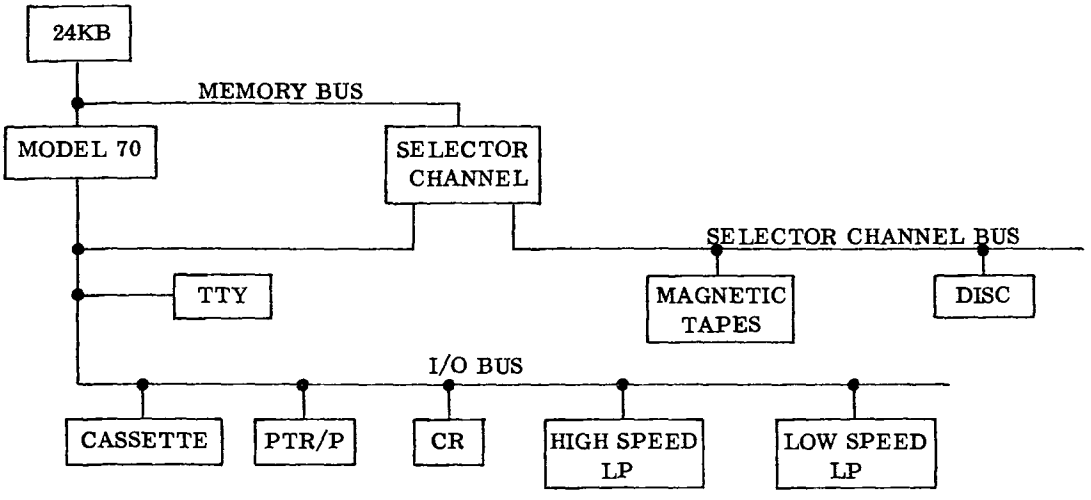


Figure 1-3. Typical DOS Configuration

Other devices supported by DOS are:

1 High Speed Paper Tape Reader or Reader/Punch

1 Card Reader

2 Line Printers (high speed and/or low speed printers)

4 9-Track Magnetic Tapes (requires Selector Channel)

3 Additional Disc Files on the same disc controller

4 INTERTAPE Cassette units

1 Additional Teletype

Refer to Chapter 5 for System Generation (SYSGEN) and configuration details.

NOTES

1. In a Model 50 system generation, DOS provides trap routines for performing RR format multiply and divide instructions (X'0C' and X'0D'). These instructions are normally illegal on a Model 50 Processor.

2. In a Model 74 system generation, DOS allows drivers within DOS to perform the list instruction ATL SPTR, LIOTRM (X'6470',X'035A').

1.4.2 Disc Allocation

DOS organizes a disc pack as follows:

Cylinder 0 - DOS itself is made resident on Cylinder 0, and can be loaded by the RTOS Bulk Storage Bootstrap Loader, 07-046.

Cylinder 1 - The DOS file directory is kept here, and is updated by the DOS supervisor.

Cylinders 2 to 202 - (404 for double density drives) - These cylinders are available for allocation to named user files.

1.4.3 Security and Protection

User disc files may be protected against both unwanted writes and reads. For more complete protection, files may be assigned names with non-printing characters.

1.4.4 Relationship to other Operating Systems

DOS is one of several operating systems offered by INTERDATA which include the BOSS family of operating systems and the Real Time Operating System (RTOS). Table 1-1 shows a comparison of data and features on INTERDATA operating systems.

TABLE 1-1. OPERATING SYSTEM COMPARISON

	BOSS/4B	BOSS/4A	BOSS	DOS	RTOS
PROGRAM NUMBER	03-021	03-020	03-019	03-022	03-017
REFERENCE DOCUMENT	29-216	29-216	29-292	29-293	29-240
CPU	all except 3	all except 3	5,50,70, 74,80	5,70,74, 50,80	5,70, 80
I/O METHOD	WAIT	INTERRUPT CONTROLLED	WAIT	INTERRUPT CONTROLLED	INTERRUPT CONTROLLED
BULK STORAGE	NOT SUPPORTED	NOT SUPPORTED	YES	YES (DISC FILE MANAGEMENT)	YES
SIZE	3KB	4.5KB	SOURCE SYSGENABLE 2.5-4.5KB	SOURCE SYSGENABLE 4-10KB	OBJECT SYSGENABLE 16KB MIN.

BOSS/4B and BOSS/4A contain similar basic features, differing mainly in the I/O handling procedure. BOSS, an outgrowth of and replacement for BOSS/4B, includes disc and drum support. RTOS is a comprehensive Real Time Operating System which includes multiprogramming capability among other extensive features. DOS replaces BOSS/4A as an interrupt driven system.

CHAPTER 2
DOS OPERATIONS

2.1 INTRODUCTION

During the period of program development, it is desirable to work with a communicative system that allows and encourages operator intervention. The ability for operator communication is necessary in a system used for program development, report generation, or data reduction. To meet these needs, DOS provides a comprehensive set of operator commands in addition to its other features.

2.2 LOADING AND STARTING THE SYSTEM

Loading DOS for the first time, or reloading DOS from a device other than disc, requires the use of either the General Loader, INTERDATA Program 06-025, or the Relocating Loader, INTERDATA Program 06-024. Either of these programs can be used to load the DOS object program produced by the System Generation process described in Chapter 5. DOS is an absolute program that includes all the standard DOS modules and any user written modules required for the particular system. Once this program has been loaded without error, the message "DOS" is printed on the Teletype and DOS is ready to receive operator command input. All logical units are assigned to the system Teletype.

2.2.1 Re-Starting at SYSGO

To re-start at SYSGO, address location X'02D0', place the Processor in the Run mode, and depress the Execute switch. This activates an initialize routine that performs the following actions.

1. It does a non-destructive top of memory search and sets an address in the memory map to indicate that all memory above the top of DOS is available for loading user programs.
2. It disables the Memory Protect Controller if one is in the system.
3. It places the console Teletype in the Read mode, after logging the message "DOS", to indicate that the system is ready. All logical unit assignments remain unchanged.

2.2.2 Loading from Bulk Storage Devices

Once DOS has been initially loaded and written to a disc, via a PACK initialize command, the core image copy can be reloaded with the RTOS Bulk Storage Bootstrap Program 07-046. This program reads the core image of DOS into memory and starts the system at SYSGO. This loader is loaded with the 50 Sequence. Location X'007A' must contain the disc file device number.

2.3 PROGRAM INTERRUPTS AND ERRORS

While user programs are running they may inadvertently generate error conditions that require some action by the system. The action taken by the system depends on the severity of the error condition. I/O device errors may also occur. When they do, the system does not log a message. Instead, the executive passes device status to the calling program and leaves it to the program to output a message.

When a memory parity or power failure occurs, DOS logs the message:

MM xxxxxxxx

where xxxxxxxx is the PSW where the parity error occurred. Once the message is logged, DOS is ready to receive operator command input. An interrupted user program may be resumed by entering CONTINUE.

NOTES

1. A program interrupted by a power failure may not be able to be restarted if I/O was in progress at the time of the power failure.

2. On systems without the AUTO RESTART option, the operator is required to depress the EXECUTE (RUN) switch to enter DOS command mode.

2.3.1 I/O Device Errors

When an I/O device error occurs, the system returns a halfword of error information to the program. The first byte of this halfword is an encoded error status; the second byte is the device address. Many programs log a message to the console device that includes the error condition and device address, such as:

I/O ERR xxpa

where xx is the error status and pa is the device address. The error status conditions are:

1. X'A0' - Device unavailable. Check to insure the device is on-line and functioning properly. If the device address is that of a non-existent device, check the logical unit assignments.

2. X'C0' - Illegal operation. Check the logical unit assignments. This type of error occurs if, for instance, a Write operation is attempted on a read only device.

3. X'90' - End of medium. The action to be taken depends on the program. For example, it may mean mounting a new tape.

4. X'84' - Unrecoverable data error; i.e., parity fail. Action depends on the type of program. It may be enough to reassign logical units.

5. X'88' - End of file. On magnetic tape devices or disc, the program has read through an End of File Mark. Action to be taken depends on the program.

2.3.2 Privileged or Illegal Instruction

If a program attempts to execute an instruction for which the operation code is illegal or privileged, the system responds by aborting the program and logging the message:

II xxxxxxxx

where xxxxxxxx is the PSW where the instruction occurred. This message also occurs if a user program attempts a supervisor call with an invalid logical unit number.

2.3.3 Arithmetic Fault Interrupts

There are two types of arithmetic fault interrupts, fixed point and floating point. DOS logs one of the following messages and continues the interrupted program at the next instruction.

Fixed Point Divide Fault:

DF xxxxxxxx

Floating Point Arithmetic Fault:

FD xxxxxxxx

where xxxxxxxx is the interrupted PSW.

2.4 OPERATOR COMMAND PROCESSOR

The operator command Processor (supervisor) interprets and acts upon directives received from the operator through the Teletype or other system input devices. If the system Teletype is the command input device DOS types an '*' indicating that it is ready to accept command input. Characters are read as typed and passed to the supervisor. If a back arrow is typed (←), the previous character is deleted. If a hash mark is typed (#), the current line is deleted. After 72 characters have been typed, or when a Carriage Return is recognized, the supervisor is activated. Operator commands consist of at least one command word such as:

```
START
```

Only the first two characters of the command word need be typed. The command is interpreted on the basis of these two characters and anything between them and the first blank is ignored. Some commands require one or more operands, which are separated from the command word by a single blank.

```
ASsign XXXX
```

Multiple operands must be separated by commas, for example:

```
ASsign XXXX, YYYY, ZZZZ
```

In some commands there are optional operands which come at the end of the operand string. The notation used to describe these commands is:

```
TR XXXX[, YYYY]
```

indicating that if the operand is not used, the preceding comma must also be left out. In some cases, there is more than one optional operand. These operands may be left out optionally from right to left.

```
ALlocate XXXX [, YYYY, ZZZZ, DDDD]
```

If for any reason, such as a misspelled command or incorrect operand, the supervisor cannot react to the command, it returns with the message:

```
?
```

2.4.1 Commands for Mass Storage Devices

The following commands relate to disc based files. They enable the operator to define, redefine, and examine the limit of mass storage files. Each disc pack on the system may be divided into as many as 200 (400 if double density) separate files. The files are named by the user. Each name contains from one to six printing or non-printing characters. Combining the file name with the device address of the disc pack completes the identification of the file.

ALLOCATE

The ALLOCATE command is used to reserve a named area on a particular physical disc pack. It may also be used to assign a file to a specific logical unit number at the time of allocation.

AL NNNNNN[, UPP, CCC, LLLL]

where NNNNNN is the name to be assigned to the disc file.

U is an optional logical unit to which the file is to be assigned.

PP is the physical disc pack address in hex (default value X'C6').

CCC is the decimal number of cylinders to be allocated to the file (default value 1).

LLLL is the decimal logical record length of the file in bytes (default value 108).

Note that 108 has been chosen as the default value for LLLL because program object records are 108 bytes long, and occur often in system operations.

Examples:

AL SOURCE, 3C6, 20, 80	Allocate a 20 cylinder file called SOURCE on pack C6, assign to logical unit 3 with a record length of 80.
AL OBJECT, C6, 2	Allocate a 2 cylinder file called OBJECT on pack C6 with a record length of 108.
AL OBJECT, D6	Allocate a 1 cylinder file called OBJECT on pack D6 with a record length of 108.
AL OBJECT	Allocate a 1 cylinder file called OBJECT on pack C6 with a record length of 108.

An ALLOCATE command may generate the following messages if error conditions are recognized.

NA-ERR	NNNNNN already occurs on the specified disc pack.
LU-ERR	U is invalid.
PU-ERR	PP is invalid (must be C6, D6, E6, or F6).
CY-ERR	The number of cylinders to be allocated (CCC) do not exist contiguously on the specified disc pack. (200 cylinders are available on a single density disc pack.)
RL-ERR	LLLL is larger than the SYSGEN value for disc physical record length.

Programming Note:

To calculate the number of cylinders needed for initial allocation of a file, the following approximation may be used:

$$C = \text{INT}\left(\frac{N*P}{256*D*\text{INT}(P/L)}\right) + 1$$

where C = approximate number of cylinders required.

N = number of records in the file.

L = logical record length in bytes.

P = disc physical record length (usually 256; established at system generation).

D = sector density (48 for single density disc).

(96 for double density disc).

and the function INT refers to the integer portion of the value within the parenthesis.

Beyond the initial allocation, DOS automatically allocates up to three overflow cylinders for sequential files and for random files if accesses are within three cylinders of the initial allocation. The user is not informed immediately when overflow cylinders have been allocated. The user may determine file overflow status by using the LIST command. Overflow cylinders are only allocated on Write operations, and never for files using the direct physical access method. If overflow cylinders cannot be allocated, because of the above conditions, or because no free cylinders remain, EOM status is returned to the user.

ACTIVATE

This command assigns a previously allocated named disc file to a logical unit number.

AC NNNNNN, U[PP, A]

where NNNNNN is the name of the disc file to be activated.

U is the logical unit number.

PP is the optional physical disc pack number (default value is C6).

A is an optional modifier indicating the file is to be opened at the next available unused record. This results in an append operation for sequential files. If A is not specified, the file is opened at logical record 0 (a logical rewind).

Examples:

AC SOURCE, 3D6, A	Assign file SOURCE on pack D6 to logical unit 3 using the file append feature.
AC OBJECT, 2D6	Assign file OBJECT on pack D6 to logical unit 2.
AC OBJ, 2	Assign file OBJ on pack C6 to logical unit 2.

An ACTIVATE command may generate the following messages if error conditions are recognized:

NA-ERR	NNNNNN does not exist of the specified disc pack.
LU-ERR	U is invalid.
PU-ERR	PP is invalid (must be C6, D6, E6, or F6).

Programming Note:

After activation, a file remains assigned to a logical unit until the logical unit is reassigned or the PACK command is given.

PACK

This utility command has two forms:

1. PA PP (CHANGE PACK)

where PP is the physical disc address, indicating to the system that a disc pack has been changed on drive PP. Any logical units that were previously assigned to disc files on the removed pack are now assigned to the system Teletype, thus insuring against accidental destruction of data on the new pack.

2. PA PP,A (INITIALIZE PACK)

initializes the disc pack with device address PP. The entire file directory and allocation map are zeroed (Cylinder 1), thus any previous data on the disc can no longer be referenced by the system. The initialize command also checks all cylinders to determine if the defective track bit is set. Defective cylinders are flagged as "in-use" in the cylinder allocation map, thus making them non-allocatable. In addition, initialization outputs a core image of DOS to Cylinder 0, thus making DOS available for bootstrap loading. Before using the pack initialize command on a disc pack for the first time, the pack should be formatted with Disc Test/Format Program 06-122.

If PP is not C6, D6, E6, or F6, then the message PU-ERR is logged on the console.

ATTRIBUTE

This command is used to define certain characteristics of a disc file.

AT NNNNNN, $X_1X_2X_3X_4$ [, PP]

where NNNNNN is the name of a previously allocated disc file.

$X_1X_2X_3X_4$ is the required attribute value of one to four hex digits. If one digit is present, it is assumed to be X_4. If two digits are present, they are assumed to be X_3X_4, etc.

PP is the optional physical pack number (C6 is the default value).

Description of Attributes:

Hex digit X_1 = unused

Hex digit X_2 = unused

Hex digit X_3 = 0 sequential

1 random (access method)

2 direct physical

Hex digit X_4 = 0 unprotected

1 write protected (file protection)

2 read protected

Programming Notes:

When a disc file is allocated, the default attributes are 0000. File attributes may be changed later by the ATTRIBUTE command. This command causes the attribute to be immediately changed on the disc directory. If the file has already been activated, a reactivation is not necessary to reflect the new attributes. If a SVC 1 attempts to write to a write protected file or read from a read protected file, illegal function status is returned to the caller. Attributes X_1 and X_2 may be specified for information purposes but are not presently used in system operation. Unspecified attribute digits are set to zero.

The sequential or random nature of a data transfer is determined by the SVC making the request. The sequential and random file attributes are used only to aid system efficiency; i.e., sequential Write operation to files with sequential attributes do not cause a preceding read.

When a file is designated as direct physical, two rules hold for all I/O transfers.

1. The random address supplied by the SVC is assumed to be the starting logical sector of the data transfer within the file.

2. The length of the data transfer is determined by the starting and ending addresses of the SVC call.

Examples:

AT SOURCE, 11, D6	Make file SOURCE on pack D6 a write protected, random file.
AT OBJ, 0	Unprotect file OBJ on pack C6.

The ATTRIBUTE command may generate the following messages if error conditions are recognized.

NA-ERR	NNNNNN does not exist on the specified disc pack.
AT-ERR	invalid attribute ($X_1X_2X_3X_4$ is invalid).
PU-ERR	PP is invalid (must be C6, D6, E6, or F6).

DELETE

This command removes a file from the disc pack library and releases its cylinders for use by subsequent allocations.

DE NNNNNN[, PP]

where NNNNNN must be a currently allocated file.

PP is an optional physical pack address (C6 is the default value).

NOTE

A read or write protected file may not be deleted. The attribute must be changed to remove protect status.

Examples:

DE SOURCE, D6	Delete file SOURCE on pack D6.
DE OBJ	Delete file OBJ on pack C6.

The DELETE command may generate the following messages if error conditions are recognized.

NA-ERR	NNNNNN does not exist of the specified disc pack.
AT-ERR	File NNNNNN is protected.
PU-ERR	PP is invalid (must be C6, D6, E6, or F6).

CLOSE

CL

The CLOSE command performs the following two functions.

1. If the DOS disc buffer contains any logical records that have been written to/from a user program and not physically written on the disc, then these records are written on the disc.
2. The file parameter information for any currently activated files are updated on the disc library. If a disc file is assigned to more than one logical unit, the parameter information of the highest logical unit is used to update the library.

Programming Note:

A SVC EOJ causes a close operation to occur. If a program using disc is terminated by an SVC PAUSE, a console interrupt, or any means other than a SVC EOJ, a CLOSE command may be needed to insure that all initiated disc transfers have been physically completed.

LIST

LI U[PP]

This command lists the entire file directory for physical disc device address PP (default value is C6) on a list device whose logical unit is U. The following is the heading and format for the List printout. Each file prints one line of data:

```
NAME    CYL RECL USED ATRB OV
NNNNNN CCC LLLL DDDD XXXX  O
```

where NNNNNN is the file name.

CCC is the decimal number of cylinders initially allocated.

LLLL is the decimal logical record length.

DDDD is the decimal number of records used (sequential files only); last logical record written (direct and random files).

XXXX is the hex attribute halfword.

O is the number of overflow cylinders in use.

Example:

```
LI 3C6
NAME     CYL RECL USED ATRB OV
SOURCE    15   80  2010 0001  0
OBJECT     1  108   150 0000  1
```

RUN

RU NNNNNN[,PP]

This command instructs DOS to load an object program contained on file NNNNNN on physical pack PP (default value i C6). After loading, DOS transfers to the origin of the program or the program transfer address, if specified.

The RUN command resets the loader bias to UBOT before loading.

Example:

RU FORTRN (Execute FORTRN on pack C6.)

The RUN command may generate the following messages if error conditions are recognized.

NA-ERR	NNNNNN does not exist of the specified pack.
PU-ERR	PP is invalid (must be C6, D6, E6, or F6).

2.4.2 General Commands

ASSIGN

This command establishes the logical to physical relationship required for all non-disc I/O calls in DOS. (Its disc counterpart is ACTIVATE.)

The format is:

AS UPP[,UPP,UPP,UPP]

where U is the logical unit number

PP is the physical device address in hexadecimal notation.

DOS allows logical unit Numbers 0 through F. The physical device addresses recognized by DOS are maintained in t Available Equipment Table (see Table 2-1) which is created by the system generation procedure (see Chapter 5). (T Available Equipment Table is referenced AETAB in the DOS listing.)

TABLE 2-1. TYPICAL AVAILABLE EQUIPMENT

DEVICE ADDRESS (HEXADECIMAL)	DEVICE
02	Teletype Keyboard/Printer
03	High Speed Paper Tape Reader
04	Card Reader
13	High Speed Paper Tape Reader/Punch
62	Line Printer
85	Magnetic Tape (9 Track)
95	Magnetic Tape (9 Track)
FF	Teletype Reader/Punch
00	Null Device
45	Cassette Tape
55	Cassette Tape
B6	Disc Controller

An example of the ASSIGN command is:

```
ASSIGN 213,500
```

This command establishes a relationship between logical unit 2 and physical address 13 so that all program references to logical unit 2 are directed to the DOS driver program for physical device 13, and causes logical unit 5 to be associated with the Null device which suppresses all I/O activity related to logical unit 5.

The ASSIGN command:

```
ASSIGN 01FF
```

associates logical unit 01 with the Teletype Reader/Punch device. This is not actually a separate device with physical address FF. It is a fictitious device whose use directs DOS to a special entry point in the Teletype driver to perform tape operations. This mechanism provides users with only Teletype I/O, to obtain Assembler and FORTRAN listings in which the source input is not interleaved with the list output. If a second Teletype has been SYSGENed into the system the TTY Reader/Punch device number is X'FE'.

Logical unit assignments remain in force until changed by another ASSIGN, ACTIVATE or PACK command. Logical units 0 through F may be assigned to any physical device for which there is an entry in the Available Equipment Table. If an ASSIGN command references a logical unit greater than F, or an unrecognized physical device address, DOS rejects the command and types out a question mark. More than one logical unit may be assigned to one physical device, but a single logical unit cannot be assigned to more than one physical device. If this is attempted, only the final assignment remains in force.

BIAS and LOAD

The BIAS command sets a constant in DOS (LDBIAS in listing), which is used by the DOS resident loader as the origin c relocatable object programs. Its format is:

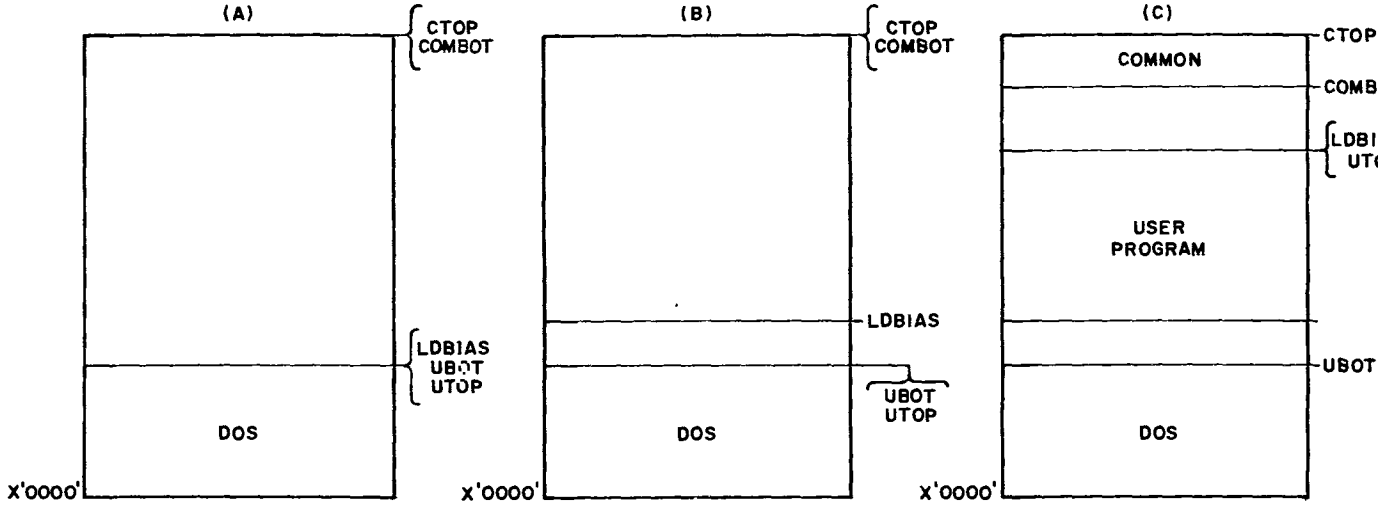
```
BI  xxxx
```

where xxxx is the desired program origin expressed in hexadecimal notation.

The LOAD command activates the DOS resident loader. Its format is:

```
LO  U
```

where U refers to a logical unit that has previously been assigned to a physical device. Figure 2-1 shows the effects o the LOAD and BIAS commands.



NOTE:
CTOP = TOP OF CORE
COMBOT = BOTTOM OF COMMON
UTOP = TOP OF USER PROGRAM
LDBIAS = LOAD BIAS
UBOT = BOTTOM OF USER CORE AREA

Figure 2-1. Core Map

Use of the BIAS Command. When DOS is first loaded or when it is reinitialized, the Load Bias Constant (LDBIAS) is s equal to the first available memory address above the top of DOS, see Figure 2-1(A). When the BIAS command is ente LDBIAS is set equal to the operand value, see Figure 2-1(B). For example:

```
BIAS  1500
```

sets LDBIAS to hexadecimal 1500. Use of the BIAS command is optional. The loader uses the current value of LDBI as the origin of relocatable programs and updates the value after each load. DOS permits a BIAS value to be set such that the user may overlay a section of DOS from location LOKUP to SUPTOP, to allow loading of large user programs If the user does not want to overlay DOS he must enter a value at or above SUPTOP.

Use of the LOAD Command. The LOAD command must be preceded by an ASSIGN or ACTIVATE command. For example:

```
ASSIGN 713 or AC OBJECT, 7
LOAD 7
```

assigns logical unit 7 to a physical file or device, and executes the resident loader with instructions to load from logical unit 7. This sequence shows the ASSIGN command immediately before the LOAD command. This is not a requirement. The only requirement is that a valid assignment be in force at the time the LOAD command is entered.

Loader Operation. The DOS resident loader loads binary object programs in loader format as generated by the Assembler, OS Library Loader and Hex Debug. Either zoned programs, labeled M08 or M09, or non-zoned programs labeled M16 or M17 can be loaded. The resident loader also loads binary data in loader format from 9 Track Magnetic Tape or disc. The loader performs all loader functions, including relocation and forward reference chaining, except the following:

1. EXTRNS and Common references (from a compiled FORTRAN program) cause the message LD ERR to be typed out. The Load operation terminates, and DOS waits for command input from the Console Teletype.

2. ENTRYs and LABELs are ignored. These items are passed over by the loader and the operator is given no indication or message.

3. Transfer addresses, if specified, are stored by the loader, but transfer to the loaded program does not occur at the completion of the load. The stored transfer address can be used in conjunction with the START command described below.

As soon as the resident loader is properly activated, it types out the message:

```
BIAS xxxx
```

where xxxx is the current value of the Load Bias Constant. During the load process, it types out the message:

```
LD ERR
EOJ
```

if the load address is less than location LOKUP in DOS. It types out the message:

```
SEQ ERR
PAUSE
```

if it detects a sequence error in the input records. If it computes a checksum that does not agree with that of the input record, it types out the message:

```
CKSM ERR
PAUSE
```

Following either sequence or checksum errors, the operator can reposition the input device, if paper tape, and resume the Load operation by typing CONTINUE. If the calculated load address at any time exceeds the top of core (CTOP in Figure 2-1), the message:

```
MEM FULL
EOJ
```

is typed out. If an I/O error occurs during the load, the loader types out the message:

```
I/O ERR xxxx
PAUSE
```

where xxxx is the error status and device address (see Chapter 3). The I/O can be retried by typing CONTINUE. At the end of each load, the loader types out the message:

```
END xxxx
EOJ
```

where xxxx is the adjusted value of the Load Bias (LDBIAS), see Figure 2-1(C).

BIAS commands with no operand field or with an operand less than location LOKUP in DOS are rejected with a question mark. For especially large user programs the user has the means of overlaying several utility routines near the top of DOS, from location LOKUP to the top of DOS (UBOT). The user should refer to the DOS listing for the routines that lie between location LOKUP and location UBOT. If the user loads a program below location SUPTOP, he must reload DOS before using any commands in Section 2.4.1.

START

The START command causes DOS to transfer control to a previously loaded user program. The format is:

```
ST xxxx
```

where xxxx is the hexadecimal address of the first instruction to be executed.

The operand field (starting address) is optional in the START command. If it is omitted, DOS starts the program at the transfer address specified on loading. If no transfer address has been specified, DOS transfers control to the instructio located at the first halfword above DOS (UBOT in listing). When the user program is started, all internal and external interrupts are enabled. The following table describes the PSW bits of the user program state.

BIT	MEANING	VALUE
0	WAIT STATE	0
1	EXTERNAL INTERRUPT ENABLE	1
2	MACHINE MALFUNCTION INTERRUPT ENABLE	1
3	FIXED POINT DIVIDE FAULT INTERRUPT ENABLE	1
4	AUTOMATIC I/O SERVICE ENABLE	1
5	FLOATING POINT ARITHMETIC FAULT INTERRUPT ENABLE	1
6	CHANNEL TERMINATION INTERRUPT ENABLE	1
7	PROTECT MODE	1
8-15		0

The starting address specified by the START command must not be less than the value of location LOKUP in DOS (except zero, which is interpreted as "no operand"). If the starting address is less than LOKUP, DOS rejects the command and types out a question mark.

@ (At sign)

The @ suspends a currently running user program. The format is:

```
@
```

Upon receipt of a @ command, DOS types PAUSE and suspends the user program. The user Program Status Word and registers are saved, and any I/O in progress is allowed to go to completion. The program can be restarted with the CONTINUE command described in the following paragraph. The @ command can be entered at any time after the START command and before the program terminates itself with an END OF JOB supervisor call (see Chapter 3). If the @ command is entered at any other time, it is rejected with a question mark.

CONTINUE

The CONTINUE command causes a previously suspended program to be restarted at the next instruction as indicated by the saved user Program Status Word. The format is:

```
CO
```

The CONTINUE command can be entered after a programmed halt (SVC PAUSE), a memory parity interrupt, or after the @ operator command.

OPEN, REPLACE, PLUS, and MINUS

These commands are used to examine and replace specified core locations. The format of the OPEN command is:

```
OP xxxx
```

where xxxx is the address, in hexadecimal, of a core location. DOS responds by typing out the contents of the specified location. For example:

```
OPEN 12F0          (Entered by operator to open location 12F0.)
4300               (Contents of 12F0 typed out by DOS.)
```

When DOS is loaded, the address of the currently open location is set to zero. Any subsequent OPEN command may change this address, but DOS does not reset it on initialization or restart. The format of the REPLACE command is:

```
RE xxxx
```

where xxxx is the value, in hexadecimal, that is to replace the contents of the currently open location. The REPLACE command is usually preceded by an OPEN command. For example:

```
OPEN 12F0          (Entered by operator to open location 12F0.)
4300               (Contents of 12F0 typed out by DOS.)
REPLACE 4130       (4130 replaces 4300 in location 12F0.)
```

The PLUS (+) and MINUS (-) commands have the format + and -. They are used with the OPEN and REPLACE commands to open the next location or the preceding location. For example:

```
OPEN 12F0          (Entered by operator to open location 12F0.)
4300               (Contents of 12F0 typed out by DOS.)
REPLACE 4130       (4130 replaces 4300 in location 12F0.)
+                  (Entered by operator to open location 12F2.)
13D8               (Contents of 12F2 typed out by DOS.)
OPEN 12F0          (Entered by operator to open location 12F0.)
4130               (Contents of 12F0 typed out by DOS.)
-                  (Entered by operator to open location 12EE.)
12F8               (Contents of 12EE typed out by DOS.)
```

2.4.3 I/O Control Commands

There are four operator commands which allow the operator to control magnetic tape, cassette, or disc from the system console. They all have a single operand which is the logical unit assigned to the desired physical device.

The commands are:

Write File Mark	WF	U	
Skip Forward to File Mark	FF	U	
Backspace to File Mark	BF	U	(Magnetic Tape and Cassette only; rewinds a disc file)
Rewind	RW	U	

where U is the logical unit number.

These commands are ignored if an improper device is specified. For disc files a logical record consisting of X'1313' i written as a file mark.

LIST LOGICAL UNIT

This command, which has the form:

```
LU [U]
```

which lists all the logical to physical assignments of DOS on a list device specified by logical unit U. If U is omitted tl Teletype is assumed as the list device. The output heading and format is as follows:

```
LU  PU  NAME
U   PP  NNNNNN
```

where U is the hexadecimal logical unit number.

PP is the physical device number.

NNNNNN is the file name if a disc file.

Example:

```
AS 162                          Assign logical unit 1 to the line printer.
LU 1                            List logical unit assignments on the line printer.
LU  PU  NAME
0   02
1   62
2   C6  SOURCE
3   D6  OBJECT
```

TRANSFER

The TRANSFER command is used to allow the supervisor to read command input from devices other than the Teletype (i.e., job control batch input). It also allows the user to log input commands on a list device if desired. The TRANSFER command has the form:

TR $[U_1, U_2]$

where U_1 is the logical unit of the new command input device and U_2 is the logical unit of the command log device. If U_2 is omitted, no logging occurs. If U_1 is omitted, command input control is returned to the Console Teletype.

Examples:

```
AS  204,162                     Assign the card reader and the line printer.
TR  2,1                         Transfer to card reader, log on line printer.
TR  2                           Stop logging.
TR                              Transfer back to TTY.
```

Note that several conditions cause control to be unconditionally returned to the Teletype:

1. An error in the command input statement.
2. An I/O error on any device involved in the command.
3. A SVC PAUSE.
4. Re-starting DOS at SYSGO.

U_1 may be assigned to any input device, while U_2 may be assigned to any output device.

Note that a command input logging feature has been added to the OS Library Loader (03-030) which corresponds to the log feature of the TRANSFER command. The Library Loader accepts the command:

LG XUU

where UU is the logical unit of the log device. The digit X is used to suppress the message 'LOADER' after each command in cases where commands are not being read from the Teletype. If X=0 the 'LOADER' message is not suppressed; if X=1 the 'LOADER' message is suppressed.

All Loader commands read from the Loader command input unit (logical unit 5) are logged on the specified logical unit. If U=0, then no logging will occur.

Examples:

DOS Commands	AS 504,362	Assign card reader and line printer.
	TR 5,3	Transfer to the card reader.
	ST	Start loader.
LOADER Commands	LG 103	Log loader commands on printer, and suppresses 'LOADER' messag
	LG 3	Continue logging and restore 'LOADER' message.
	LG 0	Stop logging.

COPY

The COPY command allows the user to copy a file from any input device to any output device. The command is:

$$CP \quad U_1, U_2, \text{ DDDD } [,A\,]$$

where U_1 is the logical unit of the input device.

U_2 is the logical unit of the output device.

DDDD is the input logical record length (decimal).

A is an optional modifier indicating termination of the COPY operation only on recognition of a file mark (disc, magnetic tape and cassette only).

If DDDD is zero, DOS assumes the input file to be 108 byte loader format records and terminates the COPY operation after a single program has been copied (the end program record has been recognized). If DDDD is specified, DOS assumes ASCII or binary data and terminates the COPY operation on an END statement (ØØ.....ØEND). If the modifier "A" is specified, the COPY operation is terminated only by a file mark. If an output disc file does not have sufficient logical record length, then data up to the disc files record length is transferred. If DDDD is less than an input disc file's record length, the DDDD bytes are transferred.

POSSIBLE SOURCE DEVICES	POSSIBLE DESTINATION DEVICES
Card Reader	Line Printer
HSPTR	HSPTP
TTY Reader/Keyboard	TTY Punch/Printer
Magnetic Tape	Magnetic Tape
Disc	Disc
Cassette Tape	Cassette Tape

Examples:

AS 104,262,485	Assign the card reader, line printer and magnetic tape.
AC SOURCE,5	Activate disc files.
AC LIBRY,6	
AC SCRT,7	
CP 1,2,80	ASCII card to print, 80 columns.
CP 6,4,0,A	Copy entire program library on disc to tape.
CP 5,7,80	Copy ASCII disc to disc.

Note that the COPY command issues a SVC 2 GET STORAGE to obtain a buffer area. The length of the buffer is equal to the length of the logical record being copied.

POSITION

This command may be used to position a file to a subfile contained within it. The beginning of a subfile is defined by a label record.

```
**NNNNNN
```

where NNNNNN is the name of the subfile. NNNNNN may be from one to six characters, with trailing blanks.

The POSITION command is of the form:

```
PO NNNNNN, U
```

where NNNNNN is the desired name, and U is the logical unit of the device containing the subfile. The POSITION command searches from the point where the file is positioned until the subfile NNNNNN is found or until an EOF or EOM condition is encountered. If either of the later conditions occur, the message I/O ERR XXXX is logged on the Teletype. If subfile NNNNNN is found, DOS returns to command mode, and the file remains positioned at the record following the label record.

Example:

The user wishes to create a disc file (JOBCTL) containing several job control programs, each having a unique name. A card deck may appear as follows:

```
**JOBCT1
AS  104,213,362
AC  SCRT,4
RU  ASMBLR
TR  0
**JOBCT2
PA  C6,A
AL  SCRT,C6,30,80
TR  0
**JOBCT3
.
.
.
.

 END
```

After copying his card deck to file JOBCTL, the user may, for example, execute the second job control program by the following commands:

```
AC  JOBCTL,3
PO  JOBCT2,3
TR  3
```

The POSITION command may also be used for source programs, user data files, etc., and on any input device. It allows the user to group large numbers of small files on a single disc file, resulting in more efficient usage of the disc.

2.4.4 Summary of DOS Operator Commands

ALLOCATE	AL NNNNNN[,UPP,CCC,LLLL] File name, logical and disc physical unit, number of cylinders, logical record length.
ACTIVATE	AC NNNNNN,U[PP,A] File name, logical and disc physical unit, file append modifier.
PACK	PA PP[,A] Disc physical unit, pack initialize modifier.
ATTRIBUTE	AT NNNNNN,XXXX[,PP] File name, attribute halfword, disc physical unit.
DELETE	DE NNNNNN [,PP] File name, disc physical unit.
CLOSE	CL
LIST PACK	LI U[PP] List device logical unit, disc physical unit.
RUN	RU NNNNNN[,PP] File name, disc physical unit.
ASSIGN	AS UPP[,UPP,UPP,UPP] Logical unit, physical unit other than disc.
BIAS	BI XXXX Program origin location.
LOAD	LO U Load device logical unit.
START	ST [XXXX] Address of first instruction to be executed.
CONTINUE	CO
OPEN	OP XXXX Memory address to be examined.
REPLACE	RE XXXX Value to be replaced.
PLUS	+
MINUS	-
HALT	@
WRITE FILE MARK	WF U
SKIP FORWARD FILE MARK	FF U
BACKSPACE TO FILE MARK	BF U
REWIND	RW U Logical unit of desired device.
LIST UNITS	LU [U] Logical unit of list device.

TRANSFER	TR [U, U] Logical unit of input device, logical unit of log device.
COPY	CP U, U, [DDDD, A] Logical unit of input device, logical unit of output device, logical record length, file mark termination modifier.
POSITION	PO NNNNNN, U Subfile name, logical unit.

2.4.5 Summary of DOS System Messages

EXEC MESSAGES

ILLEGAL INSTRUCTION	II XXXXXXXX
PARITY/POWER FAILURE	MM XXXXXXXX
FIXED PT DIVIDE FAULT	FD XXXXXXXX
FLT PT ARITHMETIC FAULT	DF XXXXXXXX
END OF JOB	EOJ
PROGRAMMED HALT	PAUSE

RESIDENT LOADER MESSAGES

BIAS VALUE	BIAS XXXX
LOAD ERROR	LD ERR
SEQUENCE ERROR	SEQ ERR
CHECKSUM ERROR	CKSM ERR
MEMORY FULL	MEM FULL
INPUT/OUTPUT ERROR	I/O ERR XXXX
END PROGRAM ADDRESS	END XXXX

SUPERVISOR MESSAGES

INPUT/OUTPUT ERROR	I/O ERR XXXX
COMMAND INPUT ERROR	?
FILE NAME ERROR	NA-ERR
LOGICAL UNIT ERROR	LU-ERR
PHYSICAL UNIT ERROR	PU-ERR
ATTRIBUTE ERROR	AT-ERR
NUMBER OF CYLINDERS ERROR	CY-ERR
RECORD LENGTH ERROR	RL-ERR
READY FOR COMMAND INPUT	DOS *

CHAPTER 3
APPLICATION PROGRAMMING

3.1 PROGRAMMING CONVENTIONS

Programs written to execute within the framework of DOS must adhere to certain established conventions to preserve the integrity of the system and to make the best use of the I/O and other supervisor services performed by DOS.

3.1.1 Privileged Instructions

The major constraint on user programming deals with the use of instructions that affect the status of the system. Because DOS controls all I/O and interrupts, user programs must not issue any instructions related to I/O functions or any instructions that change the current program status. All user programs run with external interrupts enabled, but selected internal interrupts may be disabled by a special request to the system. Table 3-1 lists the Privileged instructions. Use of these instructions results in an illegal instruction interrupt.

TABLE 3-1. PRIVILEGED INSTRUCTIONS

LPSW	SINT
EPSR	AL
WBR	WB
RBR	RB
WHR	WH
RHR	RH
WDR	WD
RDR	RD
SSR	SS
OCR	OC
AIR	AI

3.2 SUPERVISOR CALL

User programs must have a means of accessing peripheral devices and, in general, of communicating with the system in a dignified manner. INTERDATA's answer is the supervisor call interrupt, an interrupt generated by the task itself whenever it executes a Supervisor Call instruction. The format of the Supervisor Call instruction is:

```
SVC  R1,A(X2)
```

The hardware interprets the R1 field as a value rather than a register designation. It uses this value to index to 1 of 16 unique locations for servicing the interrupt. The A(X2) field, an address, is saved in a special location from which the servicing routine can retrieve it. DOS does not use all of the 16 possible variations on the Supervisor Call instruction. It recognizes only six, and uses codes in the location specified by the A(X2) field to achieve further variations. Those not recognized are treated as illegal instructions. A summary of DOS supervisor calls follows:

SVC	1,A(X2)	INPUT/OUTPUT OPERATIONS
SVC	2,A(X2)	CODE 1 - PAUSE
		CODE 2 - GET STORAGE
		CODE 3 - RELEASE STORAGE
		CODE 4 - SET STATUS
		CODE 5 - FETCH POINTER
		CODE 6 - UNPACK
		CODE 7 - LOG MESSAGE
SVC	3,0	END OF JOB
SVC	4,A(X2)	EXECUTE JOB CONTROL STATEMENT
SVC	5,A(X2)	FETCH OVERLAY
SVC	6,A(X2)	CALL A USER PROGRAM

Assigning these functions to the various supervisor calls is arbitrary. BOSS uses SVC 1 for I/O, SVC 3 for End of Job, and SVC 2 for all other supervisory services. These calls are retained in DOS for the sake of compatibility. The other calls were given unique SVC numbers to make them easier to use and easier to remember.

3.3 INPUT/OUTPUT PROGRAMMING

All supervisor calls that initiate operations on peripheral devices have two things in common. First, the first operand of the Supervisor Call instruction has a value of one; and second, the parameter block pointed to by the second operand consists of at least a function code and a logical unit, for example:

```
        SVC     1,PARBLK        I/O SUPERVISOR CALL
                .
                .
                .
PARBLK  DC      X'xxxx'         FUNCTION CODE AND LOGICAL UNIT
                .               ADDITIONAL
                .               PARAMETERS
                .               AS REQUIRED
```

The logical unit number, which occupies the second byte of the 'FUNCTION CODE LOGICAL UNIT' halfword, identifies, through the logical unit table, the peripheral device involved in the requested operation. Since DOS allows up to 16 logical units, this byte may have any value between X'00' and the number of logical units specified at SYSGEN time.

If an invalid logical unit is specified, the illegal instruction message II xxxxxxxx is logged and the program is suspended. (xxxxxxxx=PSW of the invalid supervisor call.)

The first byte of the 'FUNCTION CODE AND LOGICAL UNIT' halfword contains a value that, upon interpretation by the system, specifies the type of I/O operation. The most general interpretation of this byte hinges on the condition of its first bit, Bit 0. Setting this bit (Bit 0=1) indicates to the system that the program is requesting a control operation, such as Rewind, Backspace, or Write File Mark. Since these operations are meaningful only when applied to devices capable of performing the requested action, explanation of the control function is deferred to the sections on individual device characteristics. It is sufficient to say that, if a control function is requested of a device incapable of responding, the request is ignored.

If Bit 0 of the function byte is reset (Bit 0=0), the system infers that the program is requesting a data transfer. The encoded value of Bits 1 through 6 of the function byte indicates the type of transfer (read or write), and the modifying options selected by the program. Setting Bit 1 of the function byte indicates a Read operation. Setting Bit 2, a Write operation. Bits 3 through 5 indicate the optional modifiers chosen. Bits 6 and 7 are not used, and should always be zero. The optional modifiers offered are: ASCII or binary, wait or proceed, and random or sequential. The following list shows the various operations and options together with their binary and hexadecimal values.

OPERATION/OPTION	BINARY	HEXADECIMAL
Read	0100 0000	X'40'
Write	0010 0000	X'20'
Binary	0001 0000	X'10'
ASCII	0000 0000	X'00'
Wait	0000 1000	X'08'
Proceed	0000 0000	X'00'
Random	0000 0100	X'04'
Sequential	0000 0000	X'00'

In the binary and hexadecimal values already given, it is assumed that all bits in the function code other than the one that identifies the operation or option are zero. This was done to show the cumulative nature of these values. A coding example best illustrates this:

```
READ    EQU     X'4000'
BINARY  EQU     X'1000'
WAIT    EQU     X'0800'
LU      EQU     X'0003'
          .
          .
          .
        SVC     1,PARBLK            I/O SUPERVISOR CALL
          .
          .
          .
PARBLK  DC      READ+BINARY+WAIT+LU FUNCTION CODE AND LOGICAL UNIT
```

The code generated by the constant defined at PARBLK would be equal to X'4000' + X'1000' + X'0800' + X'0003' or X'5803 The binary representation of this constant is 0101 1000 0000 0011, in which Bits 1, 3 and 4 of the function code are set to indicate read binary and wait. The value of three in the low order byte indicates that the operation is to be performed on the device associated with logical unit 3.

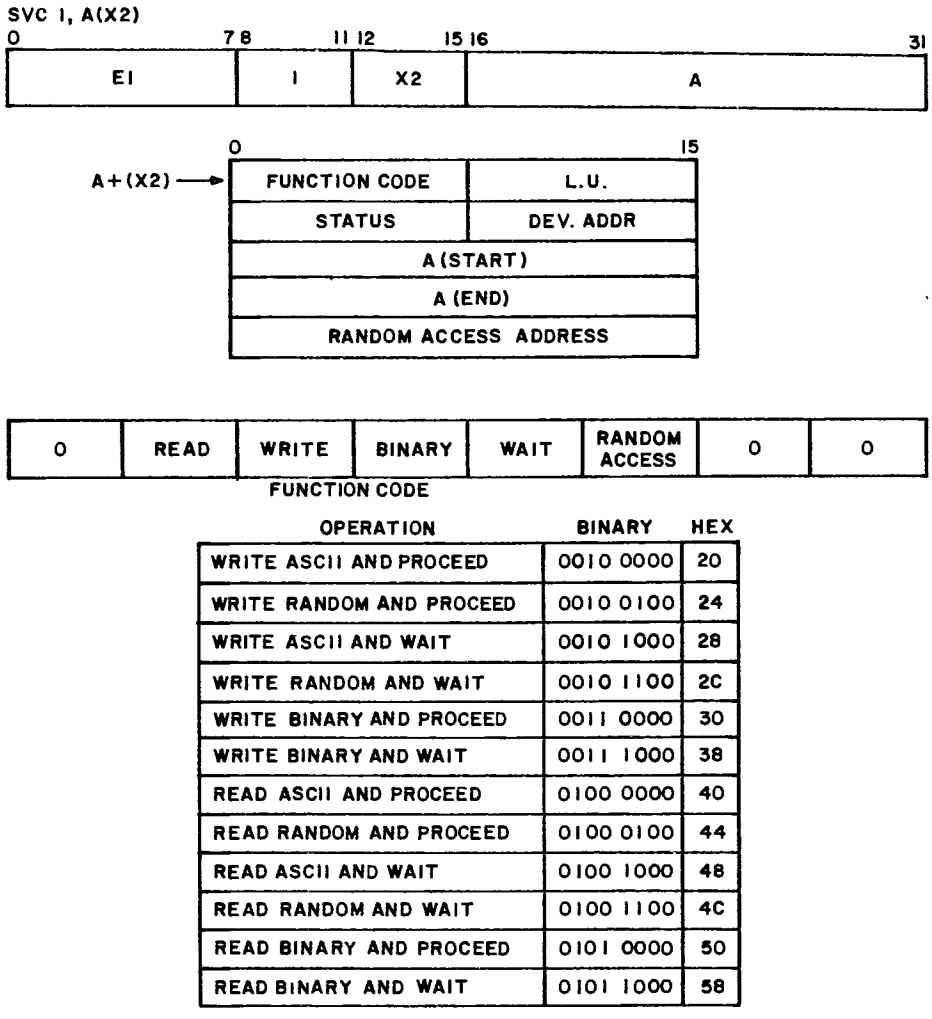
The significance of the ASCII or binary and random or sequential options is covered in the discussions on the individual devices for which they are significant. The wait or proceed option pertains to all devices, and their interpretations are as follows. I/O and wait causes the user program to go into the I/O wait state until the data transfer is complete or terminated for some reason. I/O and proceed causes the program to wait until the transfer can be started, but then allows the task to continue execution while the transfer is going on. The following programming examples show how these options may be used.

If the program specifies a function code of zero, the system returns illegal function status. If the program specifies both read and write, read is assumed, and the callers Condition Code is set to zero.

3.3.1 Examples of Data Transfers

As shown in Figure 3-1, Read and Write operations require at least three, and possibly four, additional halfwords in the parameter block of the Supervisor Call instruction. For example:

```
        SVC     1,PARBLK            I/O SUPERVISOR CALL
          .
          .
          .
PARBLK  DC      X'xxxx'             FUNCTION CODE AND LOGICAL UNIT
        DC      X'xxxx'             STATUS AND DEVICE NUMBER
        DC      A(START)            ADDRESS OF FIRST DATA BYTE
        DC      A(END)              ADDRESS OF LAST DATA BYTE
        DC      RANDOM              RANDOM ACCESS ADDRESS
          .
          .
          .
START   DS      N                   DATA BUFFER
END     EQU     *-1                 ADDRESS OF LAST BYTE
```



OPERATION	BINARY	HEX
WRITE ASCII AND PROCEED	0010 0000	20
WRITE RANDOM AND PROCEED	0010 0100	24
WRITE ASCII AND WAIT	0010 1000	28
WRITE RANDOM AND WAIT	0010 1100	2C
WRITE BINARY AND PROCEED	0011 0000	30
WRITE BINARY AND WAIT	0011 1000	38
READ ASCII AND PROCEED	0100 0000	40
READ RANDOM AND PROCEED	0100 0100	44
READ ASCII AND WAIT	0100 1000	48
READ RANDOM AND WAIT	0100 1100	4C
READ BINARY AND PROCEED	0101 0000	50
READ BINARY AND WAIT	0101 1000	58

Figure 3-1. Data Transfer Operations

The operating system uses the first byte of the 'STATUS AND DEVICE ADDRESS' halfword to return ending status to the user upon completion of the I/O operation. If the system is able to complete the transfer as requested, it stores a value of zero in this byte (and in the byte immediately following). If anything is wrong with the device before the transfer, or if anything goes wrong during the transfer, the system stores a negative value (Bit 0=1) in this byte. In addition, it stores the physical address of the device in the byte immediately following. With these items of information, the program can analyze the error condition and issue an intelligent message to the operator, who would more readily recognize a physical address rather than a logical unit number. The status returned to the program indicates any one of the following conditions.

CONDITION	BINARY	HEXADECIMAL
Illegal Function	1100 0000	X'C0'
Device Unavailable	1010 0000	X'A0'
End of Medium	1001 0000	X'90'
End of File	1000 1000	X'88'
Unrecoverable Error	1000 0100	X'84'

In general, if End of Medium, End of File, or Unrecoverable Error is indicated, then some data may have been transferred. If Illegal Function is indicated, then no data has been transferred. If Device Unavailable is indicated, and if no other bits are set, then again, no data has been transferred. Specific interpretations of these conditions, as they apply to various devices, are discussed in the following sections on individual device characteristics.

The third and fourth halfwords of the I/O parameter block specify the starting and ending addresses of the user's buffer area.

The 'RANDOM ACCESS ADDRESS' pertains only to disc and is discussed in the section on individual device characteristics.

The following examples illustrate the general programming techniques for handling data transfers. The first example uses I/O and wait.

```
        SVC     1,PARBLK        I/O SUPERVISOR CALL
        LH      R0,STAT         GET STATUS IN GENERAL REGISTER ZERO
        BM      ERROR           BRANCH IF NEGATIVE TO ERROR ROUTINE
                .
                .
                .
PARBLK  DC      X'4806'         FUNCTION CODE AND LOGICAL UNIT
*     CALLS FOR READ ASCII AND WAIT ON LOGICAL UNIT SIX
STAT    DS      2               RESERVED FOR STATUS AND DEVICE NUMBER
        DC      A(START)        STARTING ADDRESS OF BUFFER
        DC      A(END)          ENDING ADDRESS OF BUFFER
                .
                .
                .
START   DS      N               START OF BUFFER, LENGTH = N
END     EQU     *-1             FINAL BUFFER ADDRESS
```

In this example, the program requests that the system read ASCII data into the specified buffer and suspend the task until the transfer is complete. In general, specifying wait in an I/O call causes the program to go into the I/O wait state and remain there until the data transfer is complete. There are two exceptions to this rule: if the function is illegal, as would be the case if logical unit 6 were assigned to the line printer; or if the device is unavailable, as would be the case if logical unit 6 were assigned to a non-existant device, or to an inoperative device. If either of these exceptions apply, the user program resumes execution. Thus, the coding for ERROR might look like the following:

```
ERROR   THI     R0,X'4000'      TEST STATUS FOR ILLEGAL FUNCTION
        BNZ     ILLFUN          BRANCH IF ILLEGAL
        THI     R0,X'2000'      TEST STATUS FOR DEVICE UNAVAILABLE
        BNZ     DEVUNV          BRANCH IF UNAVAILABLE
        THI     R0,X'xxxx'      TEST OTHER ERROR CONDITIONS
```

Note that the program uses a define storage to reserve a location for the status and device address. This is acceptable when using I/O and wait because the contents of this halfword will always be changed by the system. Upon return to the program, it contains zero if the transfer was successful or a negative value and the device number if the transfer failed.

Another example illustrates the use of calls to read and proceed that allow processing to overlap I/O.

```
        SVC     1,PBLK1         FIRST I/O SUPERVISOR CALL FIRST BUFFER
LOOP    SVC     1,PBLK2         SECOND SUPERVISOR CALL SECOND BUFFER
        LH      R0,STAT1        CHECK STATUS OF FIRST CALL
        BM      ERROR           BRANCH ON ERROR STATUS
        BAL     Rx,PROC         BRANCH AND LINK TO PROCESS FIRST BUFFER
        SVC     1,PBLK1         REFILL FIRST BUFFER
        LH      R0,STAT2        CHECK STATUS OF SECOND CALL
        BM      ERROR           BRANCH ON ERROR STATUS
        BAL     Rx,PROC         BRANCH AND LINK PROCESS SECOND BUFFER
        B       LOOP            REFILL SECOND BUFFER
                .
                .
                .
PBLK1   DC      X'4002'         READ ASCII PROCEED LOGICAL UNIT TWO
STAT1   DS      2               RESERVED LOCATION FOR STATUS AND DEVICE NUMBER
        DC      A(START1)       START OF BUFFER NUMBER ONE
        DC      A(END1)         END OF BUFFER NUMBER ONE
PBLK2   DC      X'4002'         READ ASCII PROCEED LOGICAL UNIT TWO
STAT2   DS      2               RESERVED FOR STATUS AND DEVICE NUMBER
        DC      A(START2)       START OF SECOND BUFFER
        DC      A(END2)         END OF SECOND BUFFER
```

In this example, the program issues three supervisor calls that refer to two parameter blocks. All the calls request Read ASCII and Proceed on logical unit 2. As soon as the data transfer for the first call is started, the program resumes execution. At the second supervisor call, if the first transfer is still not complete, the task is put in the I/O wait state. When the second call can be processed, the program is again allowed to proceed. At this point, since the system was able to start a second transfer on logical unit 2, the program knows that the first transfer is complete. It checks the status and processes the data.

It then issues a third call to refill the first buffer. If the second call is not complete, the program again goes into the wait state. When the second transfer is complete, the system starts processing the third call and the program can proceed with the knowledge that the second buffer is full. It checks the status, processes the data, and loops back to repeat the sequence.

3.3.2 Teletype Input/Output

DOS supports all Teletypes as ASR Teletypes. If an ASR Teletype is used with the system, it is treated as two separate devices—a keyboard printer, and a reader/punch. The system distinguishes between these two on the basis of device assignment. Valid operations with the keyboard/printer are read, write, and wait. ASCII must always be specified on Read and Write operations. Wait and proceed are optional.

When data is read from the keyboard, it is masked to seven bit ASCII before being stored in the user's buffer. Delete characters are ignored. The transfer terminates when the buffer is full or when a Carriage Return is found in the data stream. Upon termination, the system automatically outputs a Carriage Return and Line Feed to the printer. If the transfer terminates because a Carriage Return character is found in the input stream, this character goes into the user buffer. Typing the hash mark character (#) causes the previous characters in the input stream to be ignored. The system outputs a Carriage Return and Line Feed, and the Read operation is restarted.

On output to the printer, data is sent to the printer until the user buffer is exhausted, or until a Carriage Return character is found in the data stream. On termination of the transfer, the system automatically transmits a Carriage Return and Line Feed.

Valid operations on the reader/punch are read, write, and wait. All options, ASCII or binary, wait or proceed are allowed.

In processing a read ASCII request on the paper tape reader, the system first outputs an X-ON character. It then transfers the data into the user's buffer ignoring blank tape and delete characters. Input characters are masked to seven bit ASCII before being placed in the buffer. Tapes are read in the blocked mode so data is not printed while being read. The transfer stops when the buffer is full or when a Carriage Return character is found in the data. If a Carriage Return is found, it is placed in the buffer. On termination, the system continues to advance the tape until it finds a delete character or a blank. It then outputs an X-OFF character to stop the tape.

On a request for an ASCII write to the Teletype punch, the system first outputs a RUB OUT character, a TAPE ON character, two more RUB OUT characters, and eight frames of blank tape. This is followed by the user's buffer. The transfer stops when the user's buffer is exhausted or when a Carriage Return character is found in the output stream. On termination, the system outputs a Carriage Return character (if not already output by the user), a Line Feed character, TAPE OFF, and RUBOUT. Printable ASCII characters output to the Teletype punch are printed at the same time the tape is being punched.

Binary Read operations on the Teletype reader start with an X-ON character output by the system. The tape is then advanced until the first non-zero character is found. If this character is X'F0', subsequent characters punched on the tape are read into the user's buffer until it is full. If this character is anything other than X'F0', the system assumes that the tape is punched in non-printing ASCII. In this case, the characters are read, stripped of their zone bits, and packed into the user's buffer. This process continues until the user's buffer is full. Invalid ASCII=zoned characters are ignored. After the buffer is full, the tape advances to the first blank character. On both types of binary reads, if the buffer length is shorter than the record length, overflow is lost. If the buffer length is greater than the record length, data may be garbled. The system terminates binary Read operations by sending an X-OFF character to the reader.

For binary writes on the Teletype punch, the system first sends out a RUB OUT character, TAPE ON, two more RUB OUT characters, and eight frames of blank leader. This is followed by the user's buffer. As it is output, the data is converted to non-printing ASCII characters which require two frames of punched tape for each data byte. This operatio terminates when the end of the buffer is reached. The system then outputs TAPE OFF followed by a RUB OUT characte

Two error conditions can be returned to programs using the Teletype for I/O. These are Device Unavailable (X'A0') an Illegal Operation (X'C0'). Device Unavailable is returned if the Teletype is off-line at the time of the request, or if the Break key is depressed during the transfer. The system returns Illegal Operation status if the program attempts binar operations on the keyboard/printer.

3.3.3 High Speed Paper Tape Reader Input

The High Speed Paper Tape Reader, operating under DOS, can accept read and wait commands. ASCII or binary, wait or proceed are optional with Read operations.

On ASCII input, the system ignores blank tape and delete characters. The characters are masked to seven bit ASCII before being placed in the buffer. The operation terminates when the buffer is full or if a Carriage Return character is found in the input stream. On termination, the tape continues to advance until blank tape or a delete character is found Overflow data is lost.

The system returns an error status of Device Unavailable (X'A0') if the reader is off-line at the time of the call. It returns Illegal Operation (X'C0') if a Write operation is attempted. (Write operations are legal on combined reader/punc devices that share the same device address.) The High Speed Paper Tape Reader ignores control commands. Refer t Table 3-2.

3.3.4 High Speed Paper Tape Punch Output

The High Speed Paper Tape Punch accepts write and wait commands. ASCII or binary, wait or proceed, are optional.

For ASCII output, the system first sends out eight frames of blank tape as an inter-record gap. It follows this with th contents of the buffer. The output terminates when the buffer is exhausted or if a Carriage Return is found in the data stream. The system automatically outputs a Carriage Return and Line Feed immediately following the user's data. I the user's buffer includes a Carriage Return character, this is sent to the punch followed by a system provided line fe

For binary output, the system sends out eight frames of blank tape followed immediately by the data from the user's b fer in eight bit format. The operation terminates when the buffer is exhausted.

The system returns Device Unavailable status (X'A0') to the program if the device is off-line at the time of the call, if it goes off-line during an operation. The High Speed Paper Tape Punch ignores control commands. Refer to Table 3-2.

3.3.5 Card Reader Input

The Card Reader accepts read and wait commands. ASCII or binary, wait or proceed, are optional with read comm

On ASCII input, the system converts each card column (12 bits) into one seven bit ASCII character. Illegal codes ge erate an asterisk (*) character. Each read command causes a maximum of 80 characters to be stored in the user's buffer. If the buffer is less than 80 characters in length, overflow is lost.

TABLE 3-2. LOGICAL STATUS CODES RETURNED BY DOS

DEVICE	HEX. VALUE	CAUSE
Teletype Keyboard/ Printer	00	NO ERRORS or a function was ignored.
	C0	ILLEGAL FUNCTION: an incorrect function code (i.e. write binary to the printer or read binary from the Keyboard...).
	A0	DEVICE UNAVAILABLE: Break key was depressed or an off-line condition was detected.
Teletype Reader/ Punch	00	NO ERRORS or a function was ignored.
	C0	ILLEGAL FUNCTION: as described above.
	A0	DEVICE UNAVAILABLE: as described above.
High Speed Paper Tape Reader/ Punch	00	NO ERRORS or a function was ignored.
	C0	ILLEGAL FUNCTION: a write operation was attempted on a reader only system.
	A0	DEVICE UNAVAILABLE: an off-line condition was detected.
Card Reader	00	NO ERRORS or a function was ignored.
	C0	ILLEGAL FUNCTION: a Write operation was specified.
	A0	DEVICE UNAVAILABLE: an off-line condition was detected.
Line Printer	00	NO ERRORS or a function was ignored.
	C0	ILLEGAL FUNCTIONS: a Read operation was specified or a write binary operation was attempted.
	A0	DEVICE UNAVAILABLE: either the DU bit or the EX bit was detected.
Magnetic Tape or Cassette Tape	00	NO ERRORS
	C0	ILLEGAL FUNCTION: a Write operation was specified to a write protected device.
	A0	DEVICE UNAVAILABLE: an off-line condition was detected.
	90	END-OF-MEDIUM: an end-of-tape or a beginning-of-tape condition was detected.
	88	END-OF-FILE: an end-of-file condition was detected. (file mark)
	84	UNRECOVERABLE ERROR: a parity error was detected on a Read or Write operation (ten retrys are attempted before this status is returned).
Disc	00	NO ERRORS
	C0	ILLEGAL FUNCTION: a Read or Write operation has been attempted on a file with a read or write protect attribute, or a Write operation has been attempted on a file with hardware write protect enabled.
	A0	DEVICE UNAVAILABLE: the user has attempted to use a physical file where an off-line condition, a seek-incomplete, an overrun error or a write check condition was detected
	90	END-OF-MEDIUM: file overflow condition has occurred on a direct physical file, or on a file where three overflow cylinders have already been allocated, or at a time when no unused cylinders are available for overflow allocation.
	88	END-OF-FILE: an end-of-file condition was detected. (file mark)
	84	UNRECOVERABLE ERROR: a parity error was detected on a read operation (five rereads are attempted before this status is returned) or a defective track condition has occurred.

For binary input, the system transfers twelve bits of data as read from each card column into the buffer. Two bytes of data result from each card column, with six bits of data right justified in each byte. If the buffer is less than 160 bytes in length, overflow data is lost.

Device Unavailable status is returned to the program if the Card Reader is off-line at the time of a read request. It is also returned if the device malfunctions during an operation. After receipt of Device Unavailable status, the program should notify the operator and then attempt to reread the card into the same buffer. The system returns Illegal Operation status if a program attempts to write to the card reader. The Card Reader ignores all control commands. Refer to Table 3-2.

3.3.6 Line Printer Output

The Line Printer accepts write and wait commands. ASCII must always be specified with write commands. Wait or proceed are optional.

The system outputs the contents of the user buffer. The operation terminates when the end of the buffer is reached or if a Carriage Return character is found in the data stream. A Line Printer is capable of printing 132 characters per line. If the user attempts to output more, the overflow is lost. On termination, the system causes the line to be printed on the paper and the paper to be advanced one line. Form feed and other page control characters must be contained within the user's buffer. Refer to the appropriate line printer manual for details.

Device Unavailable status is returned to the program if the printer is off-line at the time a write request is made. Illegal Operation status (X'C0') is returned if the user attempts any read or a write binary operation. Control commands are ignored. Refer to Table 3-2.

3.3.7 Magnetic Tape Input/Output

DOS supports nine track magnetic tape using the Selector Channel for data transfers. Read, Write, and wait and control operations are allowed. Read and Write operations may specify wait or proceed. The ASCII or binary option has no significance and sequential operations are always assumed.

Read requests cause eight bit data to be read from the tape directly into the user's buffer. The transfer stops when the buffer is full or when the hardware senses an end of record condition. If a parity error occurs, the system attempts to reread the record ten times before giving up. At the end of the reread operation, the tape is positioned in the inter-record gap following the defective record.

If the buffer length is less than the record length, a parity error condition occurs.

Write requests cause eight bit data to be written from memory to the tape. The operation terminates when the buffer limit is reached. If the tape is positioned at the beginning of tape marker when a write request is received, the system writes a file mark and backspaces over it before starting the Write operation. The buffer limits specified for magnetic tape transfers must start on an even byte boundary and end on an odd byte boundary.

When working with magnetic tape, all control commands are accepted. Rewind causes the tape to be positioned at the beginning of tape marker. Backspacing one record moves the tape backward over one record and leaves it positioned in the previous inter-record gap. Forward spacing one record positions the tape in the next inter-record gap. Skip forward to file mark causes the tape to skip as many records as necessary to get it to the next file mark. The tape is positioned in the inter-record gap just beyond the file mark. Skip backward to file mark causes the tape to move backward until it reaches a file mark. It goes past the file mark and stops in the preceding gap.

3.3.8 Cassette Tape Input/Output

For Cassette tape, read, write, and wait and control operations are allowed. Read and Write operations may specify wait or proceed. The ASCII or binary option has no significance and sequential operations are always assumed.

Read requests cause eight bit data to be read from the cassette directly into the user's buffer. The transfer stops when the buffer is full or when the hardware senses an end of record condition. If the buffer length is less than the record length, overflow data is lost. If a parity error occurs, the system attempts to reread the record five times before giving up. At the end of the reread operation, the cassette is positioned in the inter-record gap following the defective record.

Write requests cause eight bit data to be written from memory to the cassette. The operation terminates when the buffer limit is reached. If the cassette is positioned at the beginning of tape marker when a write request is received, the system writes a file mark and backspaces over it before starting the Write operation.

When working with cassette tape, all control commands are accepted. Rewind causes the tape to be positioned at the beginning of the tape marker. Backspacing one record moves the tape backward over one record and leaves it positioned in the previous inter-record gap. Forward spacing one record positions the cassette in the next inter-record gap. Skip forward to file mark causes the cassette to skip as many records as necessary to get it to the next file mark. The tape is positioned in the inter-record gap just beyond the file mark. Skip backward to file mark causes the tape to move backward until it reaches a file mark. It goes past the file mark and stops in the preceding gap.

3.3.9 Disc Input/Output

Each disc on the system may be divided into many named files. Individual files are treated as separate physical devices. The operator command 'ALLOCATE' defines the limits of each file on cylinder boundaries. A protect attribute can be set up for the file that prevents unauthorized programs from accessing the data in it. Read, Write, wait and control operations are allowed. Wait or proceed and random or sequential are optional.

The parameter block for disc I/O contains a halfword for random addressing. The random access address is a logical record number relative to the start of the file. (Logical sector number for files attributed as 'DIRECT PHYSICAL'.) Within each file, the records are numbered from zero, and the system makes the conversion to absolute disc address. For operations specified as sequential, the random access address is ignored.

The system maintains a current logical record address for each file. On a Read or Write operation, it increments the current address to point to the next record. This allows programs to position the current record pointer using a random access read or write and then continue the operation sequentially. For random and sequential accesses, the transfer length is limited to the logical record length of the file as allocated. For direct physical transfer, the record length may be any number of sectors as specified by the starting and ending addresses in the SVC parameter block. The starting sector may be specified by a random access address (Bit 5 set in the function code).

3.3.10 Control Operations

When the first bit of the function byte is set, (Bit 0=1), the function code is interpreted by the system as a device control operation. These operations apply to disc and magnetic tape devices. Bits 1 through 6 each designate a different operation. Bit 7 is not used and should always be zero. Valid control codes are:

BIT	OPERATION	BINARY	HEXADECIMAL
1	Rewind	1100 0000	X'C0'
2	Backspace One Record	1010 0000	X'A0'
3	Forward Space One Record	1001 0000	X'90'
4	Write File Mark	1000 1000	X'88'
5	Skip Forward to File Mark	1000 0100	X'84'
6	Backspace to File Mark	1000 0010	X'82'

The system scans the function byte from left to right. It will therefore initiate the first operation for which it finds a bit set. All control operations have an implied proceed. That is, the program goes into I/O wait only if the device is busy at the time the supervisor call is issued. Once the call can be processed, the program is allowed to continue. The following example illustrates a typical control operation (see Figure 3-2):

CONTROL OPERATIONS

	REWIND	BACKSPACE ONE RECORD	FORWARD SPACE ONE RECORD	WRITE FILE MARK	SKIP FORWARD FILE MARK	BACKSPACE FILE MARK
MAG TAPE AND CASSETTE	←		HARDWARE FUNCTIONS			→
DISC	The Current Logical Record Pointer is Set to 0	1 is subtracted from the Logical Record Pointer	1 is added to the Logical Record Pointer	Logical Record of X'1313' halfwords is written	The current logical record pointer is set to one record beyond the first detected file mark record	Same as REWIND
OTHER DEVICES	←		OPERATION IGNORED			→

```
MORE    SVC     1,PBLK1     I/O SUPERVISOR CALL
        LH      R0,STAT1    STATUS IN REGISTER ZERO
        BM      ERR1        BRANCH IF BAD
        LH      R1,START    FIRST INPUT HALFWORD IN REGISTER ONE
        CLHR    R1,R2       COMPARE WITH END INDICATOR
        BNE     MORE        MORE DATA TO INPUT
        SVC     1,PBLK2     CONTROL SUPERVISOR CALL
        LH      R0,STAT2    STATUS IN REGISTER ZERO
        BM      ERR2        BRANCH FOR ERROR RECOVERY
                .
                .
                .
PBLK1   DC      X'4804'     READ AND WAIT, LOGICAL UNIT FOUR
STAT1   DS      2           STATUS AND DEVICE ADDRESS
        DC      A(START)    BUFFER START
        DC      A(END)      BUFFER END
PBLK2   DC      X'C004'     REWIND LOGICAL UNIT FOUR
STAT2   DS      2           STATUS AND DEVICE ADDRESS
```

Figure 3-2. Control Operation Example

In this example, the program is reading records from logical unit 4. When it discovers that it has read the last record, it issues a rewind command to logical unit 4 and immediately checks the status. If the status is zero, the operation has been started, but is not necessarily complete. Subsequent operations with logical unit 4 (read, write, or control) are delayed until the rewind is complete.

3.4 SVC 2 - SERVICE FUNCTIONS (see Figure 3-3)

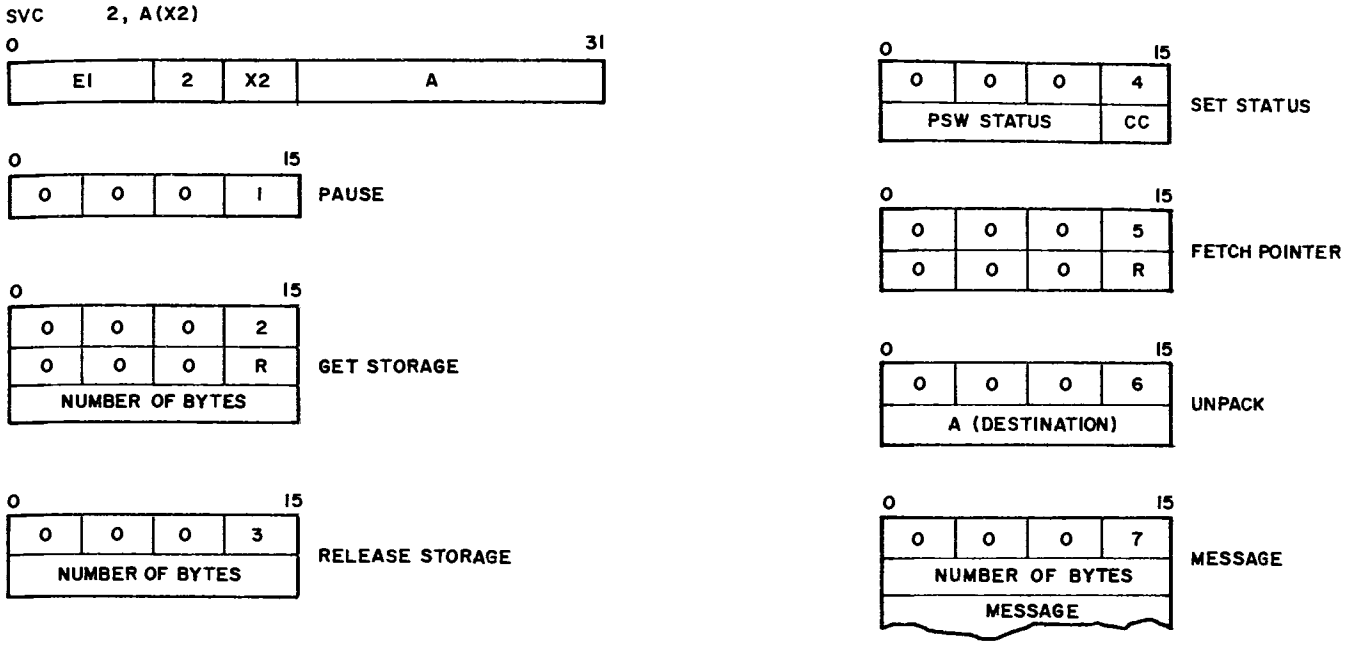


Figure 3-3. Parameter Blocks for SVC 2 Supervisor Calls

PAUSE

A program may occasionally find that it cannot continue without operator intervention. (A typical reason is trouble with a peripheral device.) The Pause supervisor call allows the calling task to suspend itself until the operator takes some action. The form of this call is:

```
        SVC     2,PARBLK        PAUSE SUPERVISOR CALL
                .
                .
                .
PARBLK  DC      X'001'          FUNCTION CODE FOR PAUSE
```

This call causes the user program to be suspended, and the word 'PAUSE' is sent to the Console Teletype. The operator, after taking whatever action may be required, can resume execution at the instruction immediately following the supervisor call by entering the CONTINUE command.

GET STORAGE

The purpose of the Get Storage supervisor call is to give a task a way to provide temporary storage locations for sub-routines it calls - especially FORTRAN Run Time Library sub-routines, which since they are reentrant, require storage outside themselves.

```
        SVC     2,PARBLK        GET STORAGE
                .
                .
                .
PARBLK  DC      X'0002'         GET STORAGE CODE
        DC      X'000x'         DESIGNATED REGISTER
        DC      N               N=NUMBER OF BYTES
```

The starting address of the area is loaded into the register specified by the second halfword of the parameter block. The third halfword of the parameter block must contain the number of bytes requested. Subsequent requests obtain new areas. In processing the Get Storage request, DOS increments UTOP, a constant initially set to the top of the user program exclusive of Common, by the amount of storage requested. It compares the new value to COMBOT, the address of the bottom of Common (or top of core when Common is not used), which is used in conjunction with FORTRAN programs. If the new value is greater than the limit, DOS terminates the program with the message:

II xxxxxxxx

where xxxxxxxx is the user Program Status Word whose Location Counter points to the supervisor call that caused the termination. Figure 3-4(A) shows the effect of a Get Storage SVC.

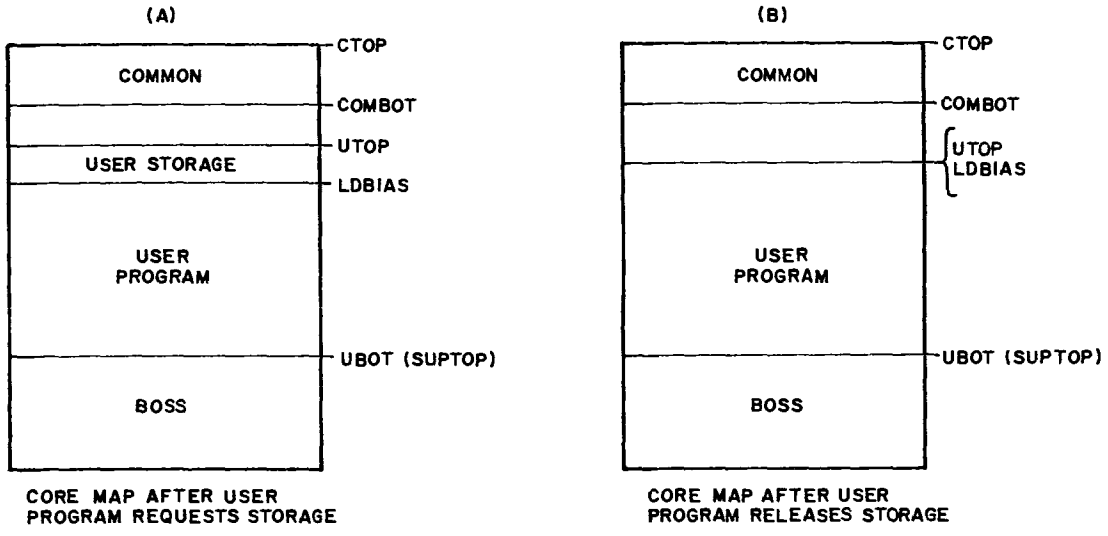


Figure 3-4. Core Map

RELEASE STORAGE

The inverse of the Get Storage call is the Release Storage call. It releases storage previously obtained. The order in which it releases core is the reverse of the order in which it was obtained. That is, the last obtained is the first released. Coding for this call is:

```
        SVC     2,PARBLK        RELEASE STORAGE SVC
          .
          .
          .
PARBLK  DC      X'0003'         RELEASE STORAGE CODE
        DC      N               N=NUMBER OF BYTES
```

The system releases the number of bytes specified by the second halfword of the parameter block, and makes those locations available for future Get Storage requests. The UTOP pointer is adjusted downward by this call.

If the value obtained is less than the value of SUPTOP, DOS terminates the program with the message:

II xxxxxxxx

where xxxxxxxx is the user Program Status Word whose Location Counter points to the supervisor call that causes the termination. Figure 3-4(B) shows the effect of a Release Storage SVC.

SET STATUS

The usual Program Status Word for application tasks is set up to allow floating point and fixed point arithmetic interrupts. At times, a task may choose not to be interrupted because of division by zero, or underflow/overflow. The Set Status supervisor call allows a task to disable and enable floating point and fixed point interrupts. The coding for this is:

```
        SVC     2,PARBLK        SET STATUS
          .
          .
          .
PARBLK  DC      X'0004'         SET STATUS CODE
        DC      X'xxxx'         NEW STATUS
```

The second halfword of the parameter block indicates how arithmetic interrupts are to be handled. The following combinations are allowed:

X'0000' Disable Floating Point and Fixed Point
X'1400' Enable Floating Point and Fixed Point
X'0400' Enable Floating Point, Disable Fixed Point
X'1000' Disable Floating Point, Enable fixed Point

Execution of this call also sets the program's Condition Code to the value of the last hexadecimal digit of the new status. A variation on this call allows the user to set only the Condition Code. It has the form:

```
        SVC     2,PARBLK        SET CONDITION CODE
          .
          .
          .
PARBLK  DC      X'8004'         SET CC CODE
        DC      X'xxxx'         NEW CONDITION CODE
```

In this call, the Condition Code of the Program Status Word is set equal to the last four bits of the second halfword of the parameter block. These calls manipulate the arithmetic interrupt enable bits and the Condition Code. They cannot be used to set or reset other bits in the Program Status Word.

FETCH POINTER

This supervisor call returns to the caller the address of a table of constants contained with DOS. It has the form:

```
        SVC     2,PARBLK        FETCH POINTER
          .
          .
          .
PARBLK  DC      X'0005'         FETCH POINTER CODE
        DC      X'000x'         REGISTER DEFINITION
```

The Fetch supervisor call places in the register specified by the second halfword of the parameter block the address of CONTAB, a table of constants in DOS. This table is summarized in Table 3-3. The table contains addresses, such as the Top of Core, which can be used by programs to communicate with the system and control storage allocation.

TABLE 3-3. CONSTANT TABLE

CONTAB + 0	CTOP	TOP OF CORE
+ 2	COMBOT	BOTTOM OF COMMON
+ 4	UTOP	USER AREA TOP
+ 6	UBOT	USER AREA BOTTOM
+ 8	LDBIAS	LOAD BIAS
+ A	XFRADR	TRANSFER ADDRESS

PARAMETER	MEANING
CTOP	Address of the last halfword within memory. This value is the highest addressable memory location.
COMBOT	Address of the first halfword within Common. This value, which is the lowest address used for Common, equals CTOP if Common is not used.
UTOP	Address of the first halfword following the user program and storage area. This value, which is the lowest available address for more storage, equals LDBIAS after a program is loaded, but no storage is allocated.
UBOT	Address of the first halfword following the DOS resident system.
LDBIAS	Address of the location to be used for loading the next relocatable program. This value which defines the first available halfword following the last program loaded, equals UBOT when no programs have been loaded, or after DOS initialization.
XFRADR	Address of execution starting point for the last program loaded. This value equals zero if the last program loaded did not specify a starting address.

UNPAK

The Unpack supervisor call converts a halfword binary constant into four ASCII hexadecimal characters. The constant to be converted must first be loaded into General Register Zero. For example:

```
        LH      R0,CONST        LOAD CONSTANT IN REGISTER ZERO
        SVC     2,PARBLK        UNPACK SUPERVISOR CALL
                .
                .
                .
PARBLK  DC      X'0006'         UNPACK CODE
        DC      A(DEST)         DESTINATION ADDRESS
                .
                .
                .
DEST    DS      4               DESTINATION
```

In this case, the constant loaded into General Register Zero is converted into four ASCII hexadecimal digits. These four bytes are then stored in consecutive locations starting at the address designated by the second halfword of the parameter block.

The Log Message supervisor call outputs a string of ASCII characters to the console device. The second halfword of the parameter block contains the number of characters, followed immediately by the message. This supervisor call should be used to output I/O error messages and operator instructions since the message always goes to the console device.

The following example of the use of the Log Message supervisor call illustrates a typical way of using several supervisor calls already described.

```
        SVC     1,PBLK1         I/O SUPERVISOR CALL
        LH      R0,STAT1        GET STATUS AND DEVICE NUMBER IN R0
        BM      ERROR           BRANCH ON ERROR
                .
                .
                .
ERROR   SVC     2,PBLK2         UNPACK CONTENTS OF REGISTER ZERO
        SVC     2,PBLK3         LOG MESSAGE
        SVC     2,PBLK4         PAUSE
        B       CONTIN          RESUME EXECUTION AFTER PAUSE
PBLK1   DC      X'4008'         READ FROM LOGICAL UNIT 8
STAT1   DS      2               STATUS AND DEVICE NUMBER
        DC      A(START)        STARTING ADDRESS OF BUFFER
        DC      END             ENDING ADDRESS OF BUFFER
PBLK2   DC      X'0006'         UNPACK CODE
        DC      A(DEST)         DESTINATION
PBLK3   DC      X'0007'         LOG MESSAGE CODE
        DC      14              LENGTH OF MESSAGE
        DC      C'I/O ERROR'    MESSAGE TEXT
DEST    DS      4               DESTINATION FOR UNPACK
PBLK4   DS      X'0001'         PAUSE
```

The parameter block for the Log Message supervisor call 'PBLK3' in the example, contains the function code, X'0007', the number of characters in the message text, in the example (14), followed immediately by the text itself. Note that in the example, the destination for the Unpack supervisor call follows the message text. The message produced by this particular sequence of coding would be:

I/O ERROR XXXX

where XXXX is the unpacked contents of General Register Zero; i.e., the logical status and physical device address returned to the program after an unsuccessful Read operation. This message would be followed by:

PAUSE

indicating that the program has suspended itself.

3.5 SVC 3 - END OF JOB

The End of Job supervisor call allows a program to terminate itself in an orderly manner. It is, following the program logic, the last executable instruction. This call has the form:

```
SVC     3,0
```

Note that there is no parameter block associated with this call. If the program issuing this call has I/O in progress at the time the call is made, the I/O is allowed to go to completion.

The End of Job SVC also causes all activated disc files to be closed. (see CLOSE command).

3.6 SVC 4 - EXECUTE COMMAND STATEMENT

This supervisor call allows the user to perform operator commands from within a running program.

The address field of the SVC 4 instruction points to an operator command statement in memory. Commands processed by SVC 4 are of the same format as those input by an operator. Commands are represented internally in 7-bit ASCII and must be terminated by either a blank (X'20') or a Carriage Return (X'0D'). If any errors are detected during the processing of a command, the incorrect command and an error message is logged on the Console Teletype and DOS retains control in the Command mode. The user may reenter the command and then enter CONTINUE to resume his program.

Example:

```
        SVC     4,PARBLK          EXECUTE OPERATOR COMMAND
PARBLK  DC      C'AC OBJECT,2 '   ACTIVATE A DISC FILE
```

While executing a SVC 4 the following conditions cause unconditional return to DOS command mode:

1. Execution of a TRANSFER command.
2. Execution of a SVC 2 PAUSE.

3.7 SVC 5 - FETCH OVERLAY

Any user program may be segmented into a trunk, or root section that resides in memory and any number of subroutines, or overlays, that are brought into memory as required. This feature is especially useful in FORTRAN applications. (See Chapter 4 for application examples.) Overlays are loaded at a location above the root section. Overlays can reference blank and labeled Common, and can reference routines in the reentrant library. Communication between the root phase and overlays is through EXTRN and ENTRY statements. The following example illustrates the use of the Fetch Overlay supervisor call:

```
        SVC     5,PARBLK        FETCH OVERLAY CALL
        BAL     RTN,OVERL1      BRANCH AND LINK TO OVERLAY
              .
              .
              .
PARBLK  DC      C'OVERL1'       SIX CHARACTER OVERLAY NAME
STAT    DC      X'0000'         STATUS AND OPTIONS
        DC      X'0004'         LOAD FROM LOGICAL UNIT 4
```

When this supervisor call is processed, the named overlay is brought into memory. The calling program must wait until the overlay is loaded.

The device assigned to specified logical unit may be any sequential input device or file capable of binary read operations. The complete overlay must be available on the device specified, (i.e., no check of the overlay name is performed).

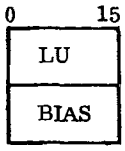
A status of zero is always returned if control is returned to the caller. If an error occurs while the overlay is being loaded, an appropriate error message is logged on the Console Teletype, and DOS returns to the command mode. The following are the possible error messages that may occur (see Chapter 2 for detailed explanation). No recovery is allowed from these conditions during overlay operations. The user should restart his program:

```
LD ERR
SEQ ERR
CKSM ERR
MEM FULL
I/O ERR XXXX
```

3.8 SVC 6 - CALL A PROGRAM

This supervisor call allows application programs to exercise considerable control over the system by calling other programs.

The parameter block for the SVC is:



where LU is the logical unit of the device from which the program is loaded. LU must have been previously assigned to a physical unit. For relocatable programs, the second halfword of the parameter block is the BIAS at which the program is loaded. If the BIAS is zero, then the program is loaded at SUPTOP. Control is passed to the loaded program at the BIAS specified in the parameter block. However, if the loaded program contained a transfer address, then control is passed to this point. Absolute programs are loaded at their origin address.

If errors occur while the program is being loaded, the appropriate error message is logged on the Console Teletype, and DOS returns control to the Command mode.

The following are the possible error messages that may occur. (See Chapter 2 for detailed explanation.)

```
LD ERR
SEQ ERR
CKSM ERR
MEM FULL
I/O ERR XXXX
```

Example:

```
        SVC     4,ASSIGN            ASSIGN A FILE
        SVC     6,PARBLK            CALL A PROGRAM
              .
              .
              .
ASSIGN  DC      C'AC OBJECT,3 '
PARBLK  DC      X'0003'             LOAD FROM LOGICAL UNIT 3
        DC      X'0000'             LOAD AT SUPTOP
```

CHAPTER 4
SYSTEM USAGE

4.1 INTRODUCTION

Efficient use of the system requires that the user take full advantage of the features inherent in DOS. This chapter provides the user with a look at various applications under DOS and thus familiarizes him with job control techniques.

4.2 MEMORY USAGE AND REQUIREMENTS

The actual size of DOS may vary from about 4KB to a maximum of 10KB depending upon the features included by the user in his System Generation procedure. DOS may be system generated as a non-disc system, similar to the INTERDATA Basic Operating System (BOSS/4A). For a full disc system the following memory is required for use of the major INTERDATA system software:

OS LIBRARY LOADER	16KB
OS ASSEMBLER	16KB
FORTRAN COMPILER	24KB

NOTE

To operate the OS Assembler on a 16KB system it is necessary to overlay various utility routines in DOS by setting the loader BIAS to location LOKUP (see DOS listing) before loading the Assembler.

4.3 SYSTEM USAGE EXAMPLES

4.3.1 Establishing a Program on Disc

The following example demontrates how the user may establish a program on a disc file for convenient access. To place the INTERDATA OS Assembler object tape on disc, Allocate a file on pack C6 as follows:

```
*AL ASMBLR,1C6
```

Note that at the same time file ASMBLR is assigned to logical unit 1.

Assign logical unit 2 to the Paper Tape Reader:

```
*AS 213
```

With the COPY command, establish the Assembler on disc:

```
*CP 2,1,0
```

To conserve disc space, two or more programs may be placed on a single disc file, and may be searched for sequentially and loaded by the OS Library Loader, or positioned via the Position command and loaded by the DOS loader.

4.3.2 Assemblies with Disc Scratch

Allocate a disc scratch file if one does not already exist.

```
*AL SCRT,C6,20,80
```

Twenty cylinders are enough for a source deck of about 3000 cards. Assume the object program will be written to a disc file called OBJECT. The following commands initiate the assembly.

```
*AS 104,362      ASSIGN THE SOURCE INPUT AND LIST DEVICES TO THE CARD
                 READER AND LINE PRINTER RESPECTIVELY
*AC SCRT,4       ASSIGN THE SCRATCH DEVICE TO FILE SCRT
*AC OBJECT,2     ASSIGN THE BINARY OUTPUT DEVICE TO FILE OBJECT
*RU ASMBLR       EXECUTE THE ASSEMBLER
```

NOTE

The Assembly OPTION card must have the SCRT option specified.

4.3.3 FORTRAN Compile and Go

Assume the FORTRAN compiler, the OS Library Loader, and the FORTRAN Run Time Library have been established on disc files FORTRN, LIBLDR and RUNTME respectively. The user wishes to compile and execute a FORTRAN source program on cards. The FORTRAN program itself reads card input from logical unit 1, outputs object code on logical unit 2, and prints output on logical unit 3.

```
*AS 104,362          ASSIGN THE COMPILER SOURCE INPUT AND LIST DEVICE TO
                     THE CARD READER AND LINE PRINTER RESPECTIVELY
*AC OBJECT,2         ASSIGN THE BINARY OUTPUT DEVICE TO DISC FILE OBJECT
*RU FORTRN           EXECUTE THE FORTRAN COMPILER
*AC RUNTME,6         ASSIGN THE RUN TIME LIBRARY
*RU LIBLDR           EXECUTE THE LIBRARY LOADER
```

The following are commands to the OS Library Loader.

```
RE 2                 REWIND LOGICAL UNIT 2
LO 2                 LOAD FORTRAN OBJECT PROGRAM
ED 6                 EDIT WITH THE RUN TIME LIBRARY
GO                   EXECUTE THE FORTRAN PROGRAM
```

4.3.4 Cataloging and Batch Operations

Using the previous example to demonstrate disc cataloging and a batch operation, allocate a disc file called JOBFTN to hold the operator commands of the previous example.

```
*AL JOBFTN,C6,1,80
```

Making the proper device assignments enables the cataloging of commands on file JOBFTN by the COPY command.

```
*AS 202              ASSIGN LOGICAL UNIT 2 TO THE TELETYPE
*AC JOBFTN,5         ASSIGN JOBFTN TO LOGICAL UNIT 5
*CP 2,5,80           TYPE OUR COMMANDS ONTO THE DISC
```

The COPY function may be terminated by typing ØEND.

To execute the job, transfer command input control to file JOBFTN. Note that JOBFTN is assigned to logical unit 5. This is necessary because the OS Library Loader, which also reads commands in this job stream, always reads command input from logical unit 5.

```
*AC JOBFTN,5
*TR 5                EXECUTE CATALOGUED JOB STREAM
```

4.4 OVERLAYS

The subject of overlays is an important one for system planning. The use of overlays can substantially reduce the memory requirements for a system. There is no limit to the number of overlays a program may have.

Overlays should not call other overlays. They should always return to the root segment (main program) and allow it to call the next overlay. If an overlay calls another overlay, the new overlay is loaded on top of (overlays) the first, and program execution resumes from the location immediately following the Supervisor Call instruction in the first overlay. This would not likely be the starting location for the new overlay.

Overlays and root segments can communicate with EXTRN and ENTRY statements. Overlays can reference subroutines that are loaded with them or loaded with the main program. Overlays can reference both blank and labeled Common.

Overlays can be very helpful in conserving memory space. However, this saving is achieved by sacrificing time. Even when disc is used as the system library, it takes time to load an overlay. For this reason, overlays should be used for processing that is not time critical.

A feature has been added to OS Library Loader 03-030 to enable the establishing of overlays. This feature is described in the Loader Descriptions Manual, Publication Number 29-231. It consists of a single command (OV), which informs the Library Loader that the subroutine about to be linked is an overlay subroutine. It is this command that origins the overlay subroutines such that they occupy a common area of memory. Overlays are established by using the OUT feature of the Library Loader. Established overlays are in absolute form.

As an example of overlay establishing procedures, assume a FORTRAN application program and two subroutines which are to be linked as overlays. The program and its subroutines have been compiled and their object programs are on paper tape. The main program calls the first overlay via logical unit 2 and the second via logical unit 3 using the IFETC routine in the Run-Time Library. We begin by allocating disc files for the main program and the overlay subroutines, and assigning them to logical units 1, 2 and 3, respectively.

```
*AL   MAINP,1C6
*AL   OVER1,2C6
*AL   OVER2,3C6
*AC   RUNTME,6          ASSIGN THE RUN TIME LIBRARY
*AS   413               ASSIGN THE PAPER TAPE READER
*RU   LIBLDR            EXECUTE THE LIBRARY LOADER
```

The following commands are read by the Library Loader.

```
OUT   1                 SET THE OUT OPTION FOR LOGICAL UNIT 1
LO    4                 LOAD THE MAIN PROGRAM
ED    6                 EDIT WITH THE RUN TIME LIBRARY
XOUT
OV                      INDICATE AN OVERLAY TO BE LINKED
OUT   2                 SET THE OUT OPTION FOR LOGICAL UNIT 2
LI    4                 LINK FIRST OVERLAY
ED    6                 EDIT WITH THE RUN TIME LIBRARY
XOUT
OV                      INDICATE ANOTHER OVERLAY IS TO BE LINKED
OUT   3                 SET THE OUT OPTION FOR LOGICAL UNIT 3
LI    4                 LINK SECOND OVERLAY
ED    6                 EDIT WITH THE RUN TIME LIBRARY
XOUT                    TERMINATE OUT OPTION
EN                      TERMINATE LIBRARY LOADER
```

The main program and its overlays have been established and may be executed from disc by the following sequence of commands.

```
*AC   OVER1,2
*AC   OVER2,3
*RU   MAINP
```

NOTE

If an overlay is to be loaded more than once by a main program, the overlay logical unit should be rewound by the main program prior to loading the overlay (i.e., in FORTRAN by a Call to subroutine REW).

4.5 DISC FILE USE

To demonstrate the use of sequential, random and direct physical access, the following sample problem is presented.

File A, which has been previously allocated by:

```
*AL FILEA,C6,1,10
```

is a sequential file that contains 200 10-byte records followed by a file mark. The first halfword of each record contains a unique binary number from 0 to 199. We wish to copy File A to another file sorting by this binary number.

Allocate File B as follows:

```
*AL FILEB,C6,1,10
```

and make FILEB random by:

```
*AT FILEB,0010
```

Activate the files:

```
*AC FILEA,1
*AC FILEB,2
```

The following sample program shows a procedure for writing File A to File B.

```
START   SVC     1,READA         READ FILE-A SEQUENTIAL
        LH      R0,READA+2      GET STATUS
        BZS     GOWRIT          NO ERRORS IF 0
        THI     R0,X'100'       CHECK FOR EOF
        BNZ     DONE            YES, DONE
        B       ERROR           NO, I/O ERROR
GOWRIT  LH      R0,BUFFER       GET BINARY NUMBER (FIRST HALFWORD OF RECORD)
        STH     R0,RANADR       MOVE TO RANDOM ADDRESS POINTER OF WRITE SVC
        SVC     1,WRITEB        WRITE FILE-B RANDOM
        LH      R0,WRITEB+2     CHECK STATUS
        BM      ERROR           I/O ERROR IF MINUS
        B       START           LOOP

READA   DC      X'4801'         READ, WAIT, LOGICAL UNIT 1
        DC      0               STATUS
        DC      BUFFER
        DC      BUFFER+9
WRITEB  DC      X'2C02'         WRITE, WAIT, RANDOM, LOGICAL UNIT 2
        DC      0               STATUS
        DC      BUFFER
        DC      BUFFER+9
RANADR  DC      0               RANDOM ADDRESS
BUFFER  DS      10
```

Now that File B has been created, let us read the entire FILEB (2000 bytes) into memory at location TABLE with one Read operation using the direct physical access method. Change the File B attribute to denote direct physical access.

```
*AT       FILEB, 0020
```

The following code performs the desired read:

```
          SVC     1,READB
READB     DC      X'4802'          READ, WAIT, LOGICAL UNIT 2
          DC      0
          DC      TABLE
          DC      TABLE+1999
TABLE     DS      2000
```

NOTE

Other access methods such as an index-sequential access method may be implemented under DOS with the use of user written subroutines. These subroutines, upon receiving control information (record keys, etc.) could perform the actual SVC 1 I/O calls to DOS.

4.6 PERFORMANCE TIMES

4.6.1 Sequential Disc File Timing Chart

This chart gives examples of the number of physical disc accesses required for transferring 1000 logical records.

PHYSICAL RECORD LENGTH

LOGICAL RECORD LENGTH	256	512	768	1024
40	167	84	53	40
80	334	167	112	84
108	500	250	143	112
150	1000	334	200	167
256	1000	500	334	250
512	——	1000	1000	500

4.6.2 Useful Time Estimates (Model 70)

I/O Driver Interrupt disable overhead time	116 microseconds
Pack Initialize time	11.5 seconds
Read time for 100 108 byte logical records (256 byte physical record length)	2.5 seconds

Examples: An overlay consisting of 100 loader format records will take approximately three seconds to load from a disc file.

To POSITN to the 100th record of an 80 byte record disc file will take approximately two seconds.

CHAPTER 5
DOS SYSTEM GENERATION AND MODIFICATION

5.1 SYSTEM GENERATION

5.1.1 Introduction

To provide the user with the flexibility of configuring DOS to conform to individual system requirements, a System Generation (SYSGEN) capability has been included. The user is provided with a complete object tape of DOS that includes all DOS options, and DOS source paper tapes. DOS System Generation is accomplished by assembling the source version of DOS with the appropriate control statements described in the following paragraphs. The user, by proper usage of these control statements, may include or delete modules of DOS during the assembly process.

DOS is designed to have a modular command and I/O structure. Because of this, all drivers with the exception of the TTY driver, and certain system commands, may be deleted. In addition, the number of available logical units or files may be selected by the user. A facility is also provided for the addition of up to two special user-written drivers. (See Section 5.1.4.)

Addition or deletion of drivers to DOS during the assembly process are made via control instructions. An assembly IF statement surrounds each module. The user must define, via EQU statements, the conditional variables for each module. The SYSGEN statements are detailed in Table 5-1. The approximate size of modules within DOS is shown in Appendix 1.

5.1.2 Conditional Assembly Procedure using the Teletype to input SYSGEN Statements

1. Load a DOS or BOSS object tape, with either the GENERAL or RELOCATING loader.

2. Load OS ASSEMBLER with the DOS or BOSS resident loader.

3. Make the desired device assignments (LU1 should be assigned to the Teletype).

4. Start the ASSEMBLER.

5. Type in an OPTION statement followed by the SYSGEN statements as described in Table 5.1. Note that the SQCHK option should not be specified. For example:

```
        OPT     PASS2
SYSTTY  EQU     2
PRINT   EQU     X'62'
        etc.
```

   All SYSGEN control statements must be included or "undefined symbol" errors result during the assembly. If new drivers are to be added, see Section 5.1.4.

6. After all control statements have been entered, the user types:

```
/PAUSE
```

   Control is now passed to DOS where the user must now assign LU1 to the paper tape reader or TTY reader, mount the DOS source paper tape, and type CONTINUE.

7. The ASSEMBLY process continues. Since the DOS source consists of five separate paper tapes, a PAUSE statement ends each tape, so that the user may mount the next tape. The user types CONTINUE to resume the assembly process. On Pass 2 (or Pass 3) of the assembly, the SYSGEN control information need not be reentered.

TABLE 5-1. SYSTEM GENERATION STATEMENTS

SYSGEN STATEMENTS:

SYSTTY	Equate to System Teletype device address (normally X'02'); do not exclude.
HSPTR	Equate to High Speed Paper Tape Reader/Punch device address (normally X'13'); equate to 0 to exclude.
CARD	Equate to Card Reader device address (normally X'04'); equate to 0 to exclude.
PRINT	Equate to Line Printer device address (normally X'62'); equate to 0 to exclude.
PRINT2	Same as above; used if a second line printer is on the system.
MAG1	Equate to Magnetic Tape #1 device address (normally X'85'); equate to 0 to exclude.
MAG2 MAG3 MAG4	Same as above; used if more than one magnetic tape is on the system.
CAS1	Equate to Cassette Tape #1 device address (normally X'45'); equate to 0 to exclude.
CAS2 CAS3 CAS4	Same as above; used if more than one cassette tape is on the system.
DSKCTL	Equate to Disc Controller device address (normally X'B6'); equate to 0 to exclude.
SELCH	Equate to Selector Channel device address (normally X'F0'); equate to 0 to exclude.
DSC1	Equate to disc file #1 device address (normally X'C6'); equate to 0 to exclude.
DSC2 DSC3 DSC4	Same as above; used if more than one disc file is on the system.
TTY2	Equate to TTY device address if more than one TTY is on the system. Equate to 0 to exclude.
SPENT1 SPENT2	Equate to device address of special device; used for adding special drivers to system; equate to 0 if no special drivers are to be added.
COPY	Equate to 1 to include the COPY utility in DOS; equate to 0 to exclude.
NUMLGU	Equate to maximum number of logical units allowed in system (must be from 1 to 16).
PHSRCL	Equate to the desired physical record in size for DOS logical disc I/O operations (must be 256, 512, 768, or 1,024). For systems with sequential processing applications a large physical record size is desirable. For random processing a small physical record size is more time efficient.
NUMCYL	Equate to the number of cylinders available on a disc file (must be 200 or 400 cylinders).
NUMSEC	Equate to the number of sectors per cylinder on disc (must be 0 for 48 sectors/cylinder, or 1 for 96 sectors/cylinder).
MOD74	Equate to 1 if the Processor is an INTERDATA Model 74, otherwise equate to 0.
MOD50	Equate to X'400' if the Processor in an INTERDATA Model 50, otherwise equate to 0.

5.1.3 Conditional Assembly using Tape or Cards to input SYSGEN Statements

1. Prepare the SYSGEN tape or deck. This tape or deck must begin with an OPTION statement and end with a PAUSE statement. The SQCHK option should not be specified.

2. Load DOS or BOSS with the RELOCATING or GENERAL Loader.

3. Load the OS ASSEMBLER. Make the desired device assignments. LU1 should be assigned to the device used to input the SYSGEN tape or deck.

4. Start the ASSEMBLER.

5. When the SYSGEN tape or deck has been processed, a PAUSE message is printed. The user should now mount the DOS source tape, reassign LU1 if necessary, and type CONTINUE.

5.1.4 Assembly Procedures to Add New Drivers

1. Follow procedures in Section 5.1.2, Steps 1-7, entering the SYSGEN statements as described in Table 5-1, including the additional optional device number SPENT1 (and SPENT2 if two drivers are being added).

2. At the end of the last DOS source tape, there is an additional pause prior to the END statement on the source tape. At this pause, the user should place his driver source module(s) into the Assembler source input device and type CONTINUE to resume the assembly process.

The following rules for user driver source must be observed:

— The entry point for driver module 1 must be or be equated to ENTRY1.

— The entry point for driver module 2 must be or be equated to ENTRY2.

— The user's source must end with the source statements

```
SUPTOP  EQU     *
        END     SYSGO
```

On second and third passes, SYSGEN EQU information need not be reentered but the user's source and END statement must be re-read in second and third passes.

5.1.5 Sample DOS System Generation

Configuration:

Model 70

One Disc Controller with 1 Disc File

Card Reader

Teletype

Line Printer

Paper Tape Reader/Punch

Two Magnetic Tape Drives

SYSGEN Control Statements

```
          OPT PASS2
SYSTTY    EQU 2          Console Teletype is device 2.
HSPTR     EQU X'13'      Paper tape reader/punch is device X'13'.
DSKCTL    EQU X'B6'      Disc controller is device X'B6'.
NUMLGU    EQU 8          Eight logical units selected.
CARD      EQU 4          Card reader is device 4.
PRINT     EQU X'62'      Line printer is device X'62'.
PRINT2    EQU 0          No second printer.
MAG1      EQU X'85'      Magnetic Tape #1 is device X'85'.
MAG2      EQU X'95'      Magnetic Tape #2 is device X'95'.
MAG3      EQU 0          Only two magnetic tapes.
MAG4      EQU 0
CAS1      EQU 0          No cassettes.
CAS2      EQU 0
CAS3      EQU 0
CAS4      EQU 0
TTY2      EQU 0          No second Teletype.
SPENT1    EQU 0          No user drivers.
SPENT2    EQU 0
PHSRCL    EQU 256        Disc physical record length is 256.
NUMCYL    EQU 200        Number of cylinders per disc is 200.
NUMSEC    EQU 0          Number of sectors per cylinder is 48.
```

```
MOD74     EQU 0                 Processor not a Model 74.
MOD50     EQU 0                 Processor not a Model 50.
SELCH     EQU X'F0'
COPY      EQU 1
DSC1      EQU X'C6'             One disc file.
DSC2      EQU 0
DSC3      EQU 0
DSC4      EQU 0
```

5.1.6 Adding User Written Drivers

Drivers for non-standard devices may be readily included in a DOS system providing these drivers are written to conform with the procedures used in the writing of standard drivers. Standard drivers are designed in such a way as to minimize the period of time during which the external interrupt line must be disabled. DOS drivers consist of at least two, and sometimes three, distinct parts. The first part, required in all drivers, is the initialize routine. In operation, this part of the driver becomes a subroutine of the Executive, which runs with external interrupts enabled. The initialize routine interprets the function code, checks buffer limits, and in general, performs any preliminary function that can be done with interrupts enabled. The second part of the driver is the interrupt service routine. This is an independent routine, although it is assembled along with other parts of the driver. The third part which, depending on the type of device and the needs of the user, may not be required, is the termination routine. Like the initialize routine, this becomes a subroutine of the executive. It performs such functions as error identification, movement of data, code conversion, and interpretation of final device status.

The user written driver routines can be added to the system, provided they use the same linkage conventions as INTERDATA supplied drivers. The DOS listing supplies several examples of driver programming. The user should refer to the DOS drivers for examples of driver linkage logic. All driver programs must be entered through the supervisor. The general flow is as follows (see Figure 5-1): (Refer to Figure 5-2 for detailed I/O logic flow.)

1. The user program issues a supervisor call for I/O.

2. The supervisor saves the user registers, establishes the logical to physical relationship, loads the driver registers with pertinent information from the Device Control Block (DCB) and branches to the driver. External Interrupts are enabled upon initial entry to the driver.

3. The driver does any initial processing and verification necessary, and performs a BALR DVR,INTRET which returns to the user program.

4. The driver now regains control (via the SINT instruction executed in subroutine SVCRET) with interrupts disabled. The user may now activate his device and perform other BALR DVR,INTRET to await device interrupts. (To exit from all resulting device interrupts, execute a BALR DVR,INTRET.)

5. When all device interrupts have been processed, the user performs the instruction ATL SPTR,LIOTRM (causing a queue termination interrupt) and exits via another BALR DVR,INTRET.

6. The driver now regains control via the queue termination interrupt, with interrupts enabled. The user may now do any termination processing, and make a final exit with the instruction BR UBSY.

As an example of driver logic, we will examine the card reader driver in a step by step manner. Refer to the DOS listing to follow the logic.

A SVC read causes entry to the driver, with external interrupts enabled, at location CRDVR. After checking for Illegal commands and bad device status, the driver executes a BALR DVR, INTRET which returns to the calling program. While returning from the SVC, the executive executes a SINT 0(DEV) which causes an external interrupt and a return to the driver at the instruction following BALR. With interrupts now disabled, the driver adjusts FBA, if necessary, and feeds a card. DVR is now loaded to point to a routine called SENSE which checks the device status on every card column interrupt. The driver branches to SENSE via BALR RTN, DVR. SENSE exits to the user via BTCR 8, INTRET if busy is high (a column not available for reading). When a card column is ready for reading, an interrupt occurs, and control is passed to SENSE (contents of DVR). Since busy is now low, SENSE executes BR RTN which, in turn, returns us to the RHR instruction in the card reader driver. After reading the column, code conversion is performed and the resulting byte is stored in the user buffer (routine HTAXIT). The BXLE CBA, INTRTN instruction returns to wait for another column interrupt if all requested columns have not been processed. If all columns have been read, it loads DVR to force a loop waiting for EOM status. If no EOM status is sensed, the BFCR 2, INTRET exits to wait for another interrupt. If EOM status is sensed, the driver performs an ATL SPTR, LIOTRM (subroutine HSRPDN) instruction which places the contents of SPTR in a list and causes a queue termination interrupt. A BALR DVR, INTRET returns from the previous external interrupt (EOM), and we return immediately following the BALR DVR, INTRET as a result of the queue termination interrupt. At this time, external interrupts are enabled, and we may perform any desired termination processing. In this case, there is none, so the driver completes the process with BR UBSY which frees the device for subsequent calls.

Each device driver requires a Device Control Block (DCB). The device control block is used to save the driver registers, and control entry to the driver from external interrupts. When user drivers are added to the system using SYSGEN statements SPENT1 and SPENT2, a DCB is automatically created for the user. All DCB's are of the form:

```
DCB     DC  0,0,0,0,0,1,0,DEVISP,DEV,DEVBSY,UBSY1,0,0,0,0,0
DEVISP  DC  0,0,0                   External interrupt PSW save area
        STM 0,RSAVE                 Save user registers
        LM  0,*-42                  Load driver registers
        BR  DVR                     Go to driver
```

DEVISP is pointed to by the Interrupt Service Table (location X'D0'). The following is a breakdown of driver registers and their uses.

REG	SYMBOLIC	CONSTANT CONTENTS	MEANING
0	DVR		Driver re-entry address whenever a driver is exited; DVR should be pointing to the return location.
1	INTRET	SVCRET or INTRTN	Address of the SVC return routine (first entry to driver) or the external interrupt return routine (subsequent entries).
2	AX2		Pointer to the SVC parameter block.
3	FCN		Contents of first halfword of SVC parameter block (function code and logical unit).
4	CBA		User buffer start address.
5	ONE	1	Constant of 1.
6	FBA		User buffer end address.
7	SPTR	A(DEVISP)	Pointer to the interrupt service entry of the DCB.
8	DEV	DEV	Device number.
9	AET	A(DEVBSY)	Pointer to the device busy flag in the available equipment table.
10	UBSY	A(UBSY1)	Pointer to the common driver termination routine.

REG	SYMBOLIC	CONSTANT CONTENTS	MEANING
11	ERRTN		Error status register.
12	ACO		General accumulator
13	AC1(RTN)		General accumulator (subroutine return address).
14	AC2		General accumulator.
15	AC3		General accumulator.

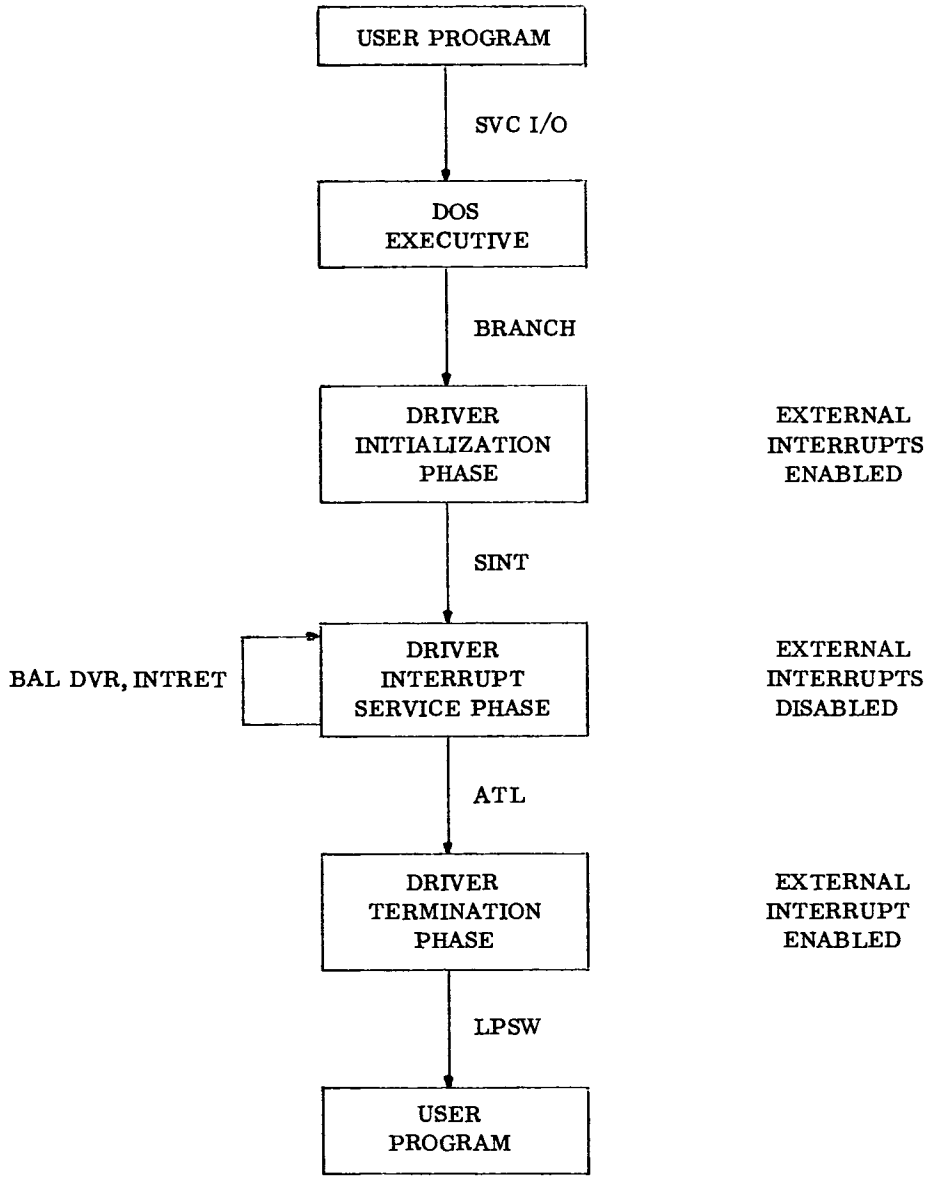


Figure 5-1. I/O Logic General Flow

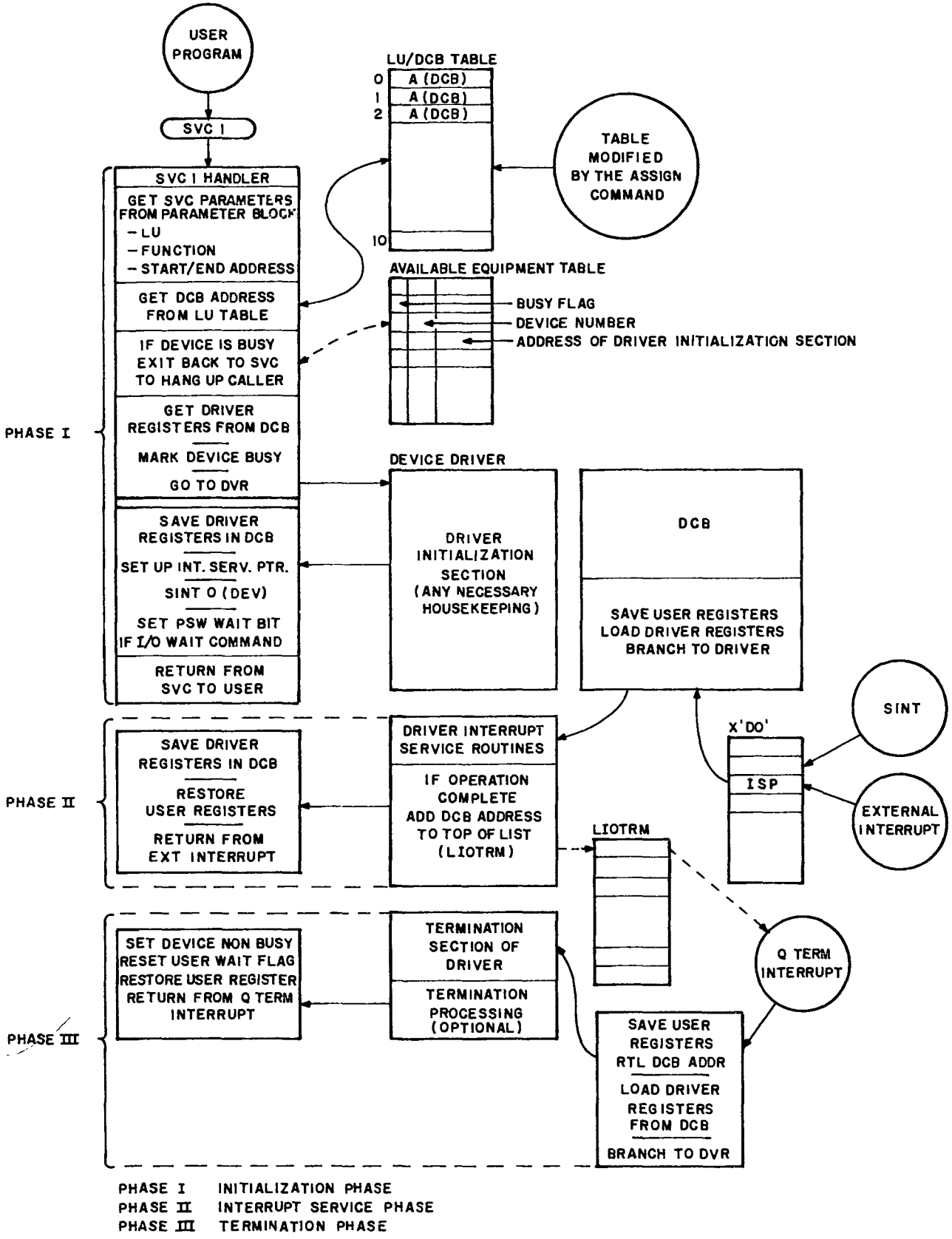


Figure 5-2. I/O Logic Detailed Flow

APPENDIX 1
APPROXIMATE NUMBER OF BYTES REQUIRED FOR OPTIONAL SYSGEN MODULES

HSPTR	171
DSKCTL	3100 (includes all disc oriented operator commands)
NUMLGU	32 per logical unit if DSKCTL specified
CARD	300
PRINT	120
MAG1	400
CAS1	400
PHSRCL	value specified in EQU statement
MOD74	64
MOD50	275
COPY	320

CUT ALONG LINE

PUBLICATION COMMENT FORM

Please use this postage-paid form to make any comments, suggestions, criticisms, etc. concerning this publication.

From _____ Date _____

Title _____ Publication Title _____

Company _____ Publication Number _____

Address _____

_____

_____

FOLD FOLD

Check the appropriate item.

Error (Page No. ___, Drawing No. _____)
Addition (Page No. ___, Drawing No. _____)
Other (Page No. ___, Drawing No. _____)

Explanation:

FOLD FOLD

Fold and Staple
No postage necessary if Mailed in U.S.A.