OS/32-ST PROGRAM REFERENCE MANUAL

INTERDATA®

OS/32-ST PROGRAM REFERENCE MANUAL

Publication Number 29-380

This is a PRELIMINARY Manual and is subject to change without notice.

PRINTED IN U.S.A. APRIL 1974

OS/32-ST

PROGRAM REFERENCE MANUAL

PREFACE

OS/32-ST is a Serial Tasking Operating System designed for the 32-Bit Architecture Processors. The reader should be familiar with the 32-Bit Series Reference Manual, Publication Number 29-365.

Other related user's manuals are:

- OS/32 User Guides, Publication Number 29-393
- OS/32 Series General Purpose Driver Manual, Publication Number 29-384
- OS/32-ST Program Configuration Manual, Publication Number 29-379
- OS/32-ST Program Logic Manual, Volume 1 (Text), Publication Number 29-381
- OS/32-ST Program Logic Manual, Volume 2 (Flow Charts), Publication Number 29-381

OS/32-ST

PROGRAM REFERENCE MANUAL

TABLE OF CONTENTS

ILLUSTRATIONS

APPENDICES

CHAPTER 1

SYSTEM DESCRIPTION

1.1 ABSTRACT

OS/32-ST is an operating system that provides system control, resource allocation and program management for single task programs on a 32-Bit Series Processor. Console operator facilities, interrupt handling, and Input/Output (I/O) servicing are built-in functions of OS/32-ST. Data file management features are provided when the system is equipped with a direct access device (disc), and consequently OS/32-ST is oriented towards a disc operating system environment. Command sequences may be executed from Command Substitution System (CSS) files so that complex command functions can be issued with one statement. Memory, device and (disc) file resources can be allocated by the console operator according to configuration and program needs. Memory and files can also be allocated by user programs. Large programs can be accommodated because OS/32-ST fully supports the ability of the Processor to directly address up to one megabyte of memory.

This manual provides the information needed for operating the system and for writing programs that run under it. More information on use of major OS/32 features can be found in the OS/32 User Guides manual, Publication Number 29-393.

Refer to the OS/32 Series General Purpose Driver Manual, Publication Number 29-384, for peripheral device driver characteristics, program documentation, and the requirements of user-written drivers.

Refer to the OS/32-ST Program Configuration Manual, Publication Number 29-379, for system generation (SYSGEN) procedures and related configuration, size and performance information. A preSYSGENed OS/32-ST STARTER program is provided which is adequate for many user needs. STARTER is also described in the OS/32-ST Program Configuration Manual, Publication Number 29-379.

Refer to the OS/32-ST Program Logic Manual, Publication Number 29-381, for the purpose of making modifications and program maintenance.

1.2 IMPORTANT FEATURES

Named Devices and Files

All peripheral devices are referred to by SYSGENable mnemonic names rather than by hardware addresses which are configuration-dependent. Direct access files are also referred to by name and assigned in the same manner as devices. File names are composed of three fields: Volume Name (the name by which a direct access volume is known to the system), File Name (intended to designate a class of files related in some way), and Extension (intended to designate variously processed forms of the same file)

Device and File Protection

Each device and file is protected by means of read and write keys which must be supplied by the user program or console operator when trying to gain access to that file or device. Keys are eight-bit numbers from X'00' to X'FF'; X'00' being unconditionally accessible and X'FF' being accessible only by executive system routines.

Device and File Allocation

Devices are defined at SYSGEN time; files may be allocated by the console operator or dynamically by user programs. Devices may be renamed and their read and write keys may be changed by the operator. Files may alternatively be renamed and have their read and write keys changed by user programs. Devices may be marked on or off-line by the operator.

Standard I/O

Input/Output (I/O) programming is facilitated by a conventional manner in which programs call the executive to perform I/O operations. I/O is device independent so that, for example, a command stream can be read from a card reader to a direct access file in exactly the same manner.

Concurrent I/O

I/O requests by programs are performed as either wait (non-overlapped) or I/O proceed (overlapped).

Direct Access Files

Direct access files may be created, named, protected, examined, and deleted. A file may be created using one of two structures, both of which support random and sequential access:

Contiguous - A closed and unbuffered organization of a fixed (when allocated) number of data blocks which are allocated as an unbroken series of block addresses.

Chained - An open-ended and buffered organization of chain-linked data blocks. Blocking and deblocking of logical records is performed automatically by the system. Logical record size is independent of physical record size.

Disc Pack Interchange

Disc packs with data files may be removed and reinserted into the system configuration because all the control information required by the system is resident on the pack.

Overlays

Programs need not be totally memory-resident. They can be segmented and overlaid from a device or file.

Console Log

The operator may transfer logging functions to a device other than the console device presumably to a line printer or to a direct access file.

Command Processing Routines

Command processing routines in OS/32-ST are available to user programs.

Command Substitution System

The Command Substitution System (CSS) is a powerful extension to the OS/32-ST Command Processor. It provides the user with the ability to establish files of commands which can be called from the console and executed in a defined sequence. In this way complex operations can be carried out by the operator with only a small number of commands. For instance, to compile, load and run a FORTRAN program, only a single command need be entered.

1.3 SYSTEM CONFIGURATION

A typical OS/32-ST hardware configuration well suited for program development is illustrated in Figure 1-1. Other peripherals are available, such as paper tape and magnetic tape equipment.

For initial loading and execution, OS/32-ST requires no support software other than the 32-Bit Relocating Loader, (Program Number 03-067). If the user has a disc in his configuration, he may establish a memory image of OS/32-ST on disc and reload rapidly with the OS/32 Bootstrap Loader, (Program Number 03-074).

Users normally make use of INTERDATA-provided OS programs, such as the CAL Assembler (Program Number 03-066), FORTRAN IV Level 1 Compiler (Program Number 03-060), and the OS/32 Library Loader (Program Number 03-065) for their program development. Several operation examples are given in Chapter 4.

Typical software configurations of OS/32-ST may be generated from a library of system object modules by using the OS/32-ST Configuration Utility Program (CUP/ST; Program Number 03-076). Various optional features and device selections may be made during this object SYSGEN process, including the removal of file management routines if the system is not configured with direct access devices. Certain options may also be selected by reassembling system source.

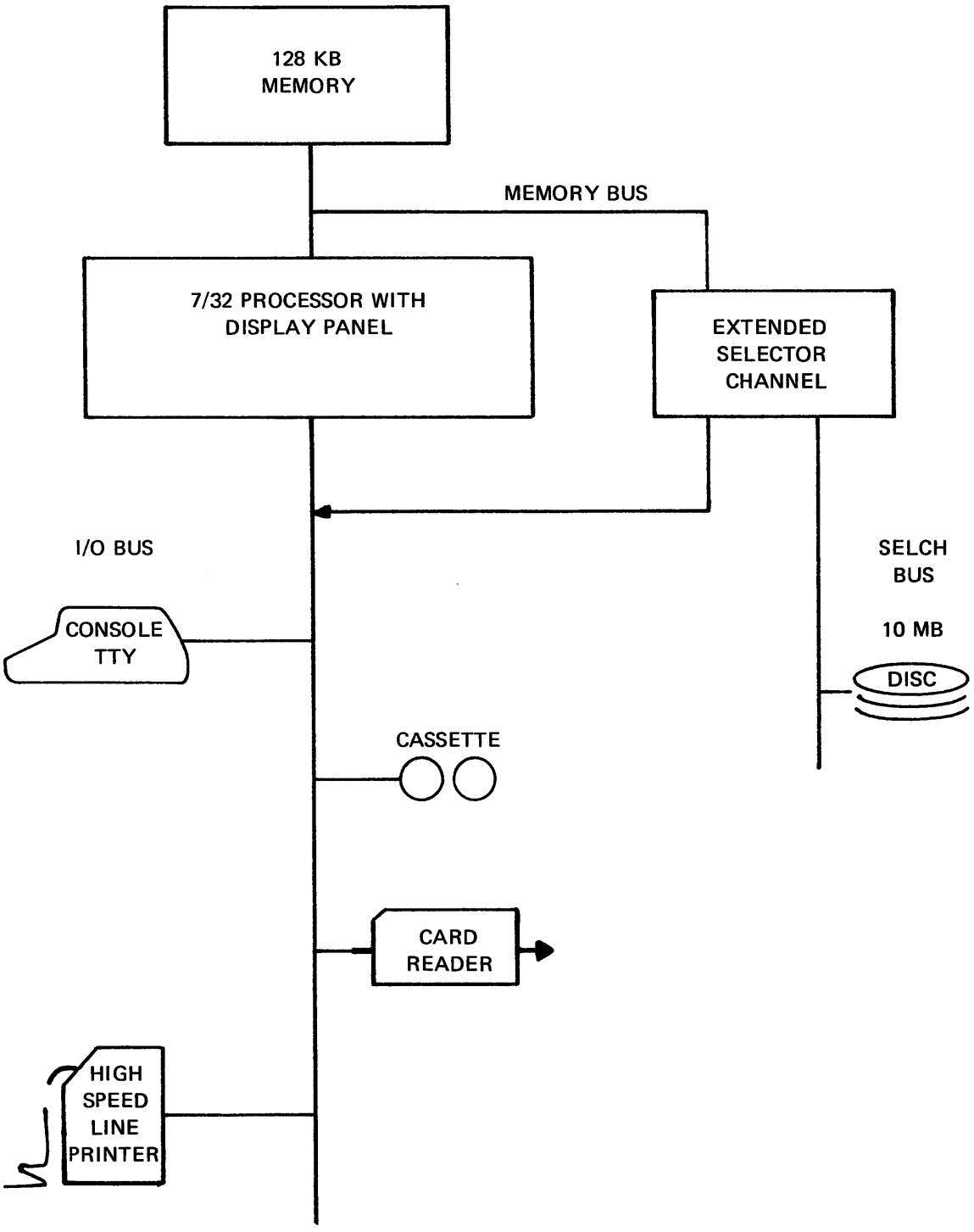


Figure 1-1. Typical OS/32-ST Configuration

Selectable SYSGEN options include:

- Inclusion of Floating Point traps
- Deletion of the Command Substitution System
- Deletion of file management support
- Deletion of Chained file support
- The name of system console device
- The name of system direct access volume
- Default system parameters
- Specification of various system/task parameters, drivers, device names, etc.

A preSYSGENed OS/32-ST STARTER program (Program Number 03-075) is provided to run CUP/ST. OS/32-ST STARTER may have general applicability, depending on user requirements. The system generation process and STARTER are described in the Program Configuration Manual, Publication Number 29-379.

For its direct-access storage medium, an OS/32-ST configuration may include one or more of several types of discs. This manual generally refers to a disc as the direct-access device. On the other hand, file management procedures in OS/32-ST are not directly dependent on a particular disc or disc controller. Consequently, any type of direct-access device may be employed if it is characterized by 256 byte sectors or can emulate 256 byte sectors and if an appropriate I/O driver is provided.

1.4 MEMORY MANAGEMENT

A memory map of an OS/32-ST configuration is illustrated in Figure 1-2. Memory address space must be contiguous for all configured memory. The Operating System occupies low memory, followed in turn by one or more user programs, user data area, unused space, and system allocated direct access File Control Blocks (FCBs) at the top.

Memory management concerns the values of certain locations as maintained by the following system parameters:

User Bottom (UBOT)	The first halfword location of user space above the operating system; it is a constant dependent on the size of the user's configured OS/32-ST (does not change by loading user programs).
User Top (UTOP)	The first halfword location above the user's program; it changes with each program loaded.
Common Top (CTOP)	The last halfword location of the user's allocated memory.
FCB Bottom (FBOT)	The first halfword location of the direct access FCB area.
Memory Top (MTOP)	The location of the last halfword of configured memory, plus two.

The way in which memory is allocated in OS/32-ST is as follows (see Figure 1-2).

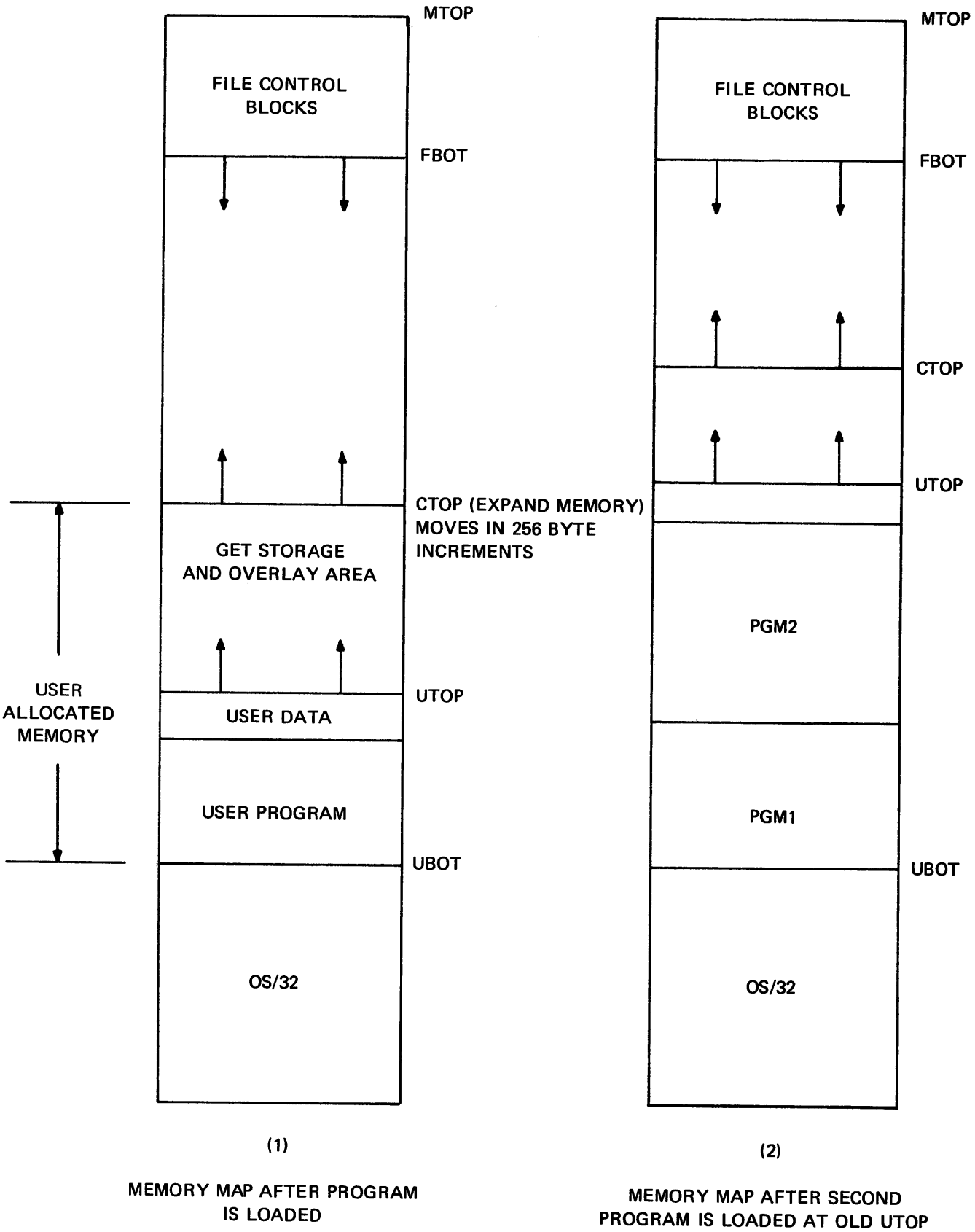


Figure 1-2. OS/32-ST Memory Map

If a load address is not specified when a program is loaded, space is made available from the value of UBOT upwards; i.e., from the bottom up. The new program top address is placed in UTOP. CTOP is then set to the next higher multiple of 256 bytes (or more depending on the SYSGENed option).

The program may obtain additional memory area by issuing Get Storage calls (say for temporary data as required by FORTRAN library routines). Get Storage calls increase UTOP. Overlays are also loaded in the area between UTOP and CTOP.

The parameters UBOT, UTOP and CTOP provide compatibility with other INTERDATA operating systems.

To load a new program without overwriting previously loaded programs, the operator must specify a load bias. He can determine the current value of UTOP and CTOP by displaying system parameters via an operator command (DISPLAY).

File management routines require FCBs in memory for active files. They contain control information and buffer space. FCBs are allocated dynamically by the system when file are assigned. Space is allocated from the top of memory down. The internal parameter FBOT is used to keep track of the current FCB area bottom. If the allocation of a new FCB would cause FBOT to overlap CTOP, the allocation request is rejected. The size of FCBs and buffers is a function of the type of files, but the operator can ascertain the value of FBOT by displaying system parameters with an operator command (DISPLAY).

When an Expand Memory Allocation request is executed, CTOP is incremented in multiples of 256 bytes. If this would cause CTOP to overlap FBOT, the allocation request is rejected. On a Contract Allocation request, CTOP is decremented. The request is considered invalid if it would cause UTOP to exceed CTOP.

When a Get Storage request is executed, UTOP is incremented. If this would cause UTOP to exceed CTOP, the request is rejected. Release Storage works in much the same way except that UTOP is decremented. Storage is not allowed to be released past UBOT.

1.5 SYSTEM STRUCTURE

As illustrated in Figure 1-3, OS/32-ST is functionally composed of five major module groupings. These are the executive, command processor, loader, I/O subsystem, and the file management package.

Running under the Operating System, the user task (user program) may consist of a single program, or it may include a main program and a number of subroutines and overlays. User programs may be resident in memory, or they may be loaded as required. The total number of programs allowed in memory at any given time is limited only by the amount of memory available, but only one user task is known to the system at any given time.

User tasks are not privileged; they are executed with the Processor's Protect bit set in the Program Status Word (PSW) so that an attempt to execute a privileged instruction causes an Illegal Instruction Interrupt. Privileged instructions are the type that perform I/O operations or change the state of the Processor. Tasks can execute in a privileged state, called Executive mode, but users should not ordinarily employ it. This capability is provided for running system programs.

As shown in Figure 1-3, there are two distinct user interfaces to OS/32-ST. One is the operator command interface that allows the operator of the console device to control the system (see Chapter 2). The other interface is at the program level;

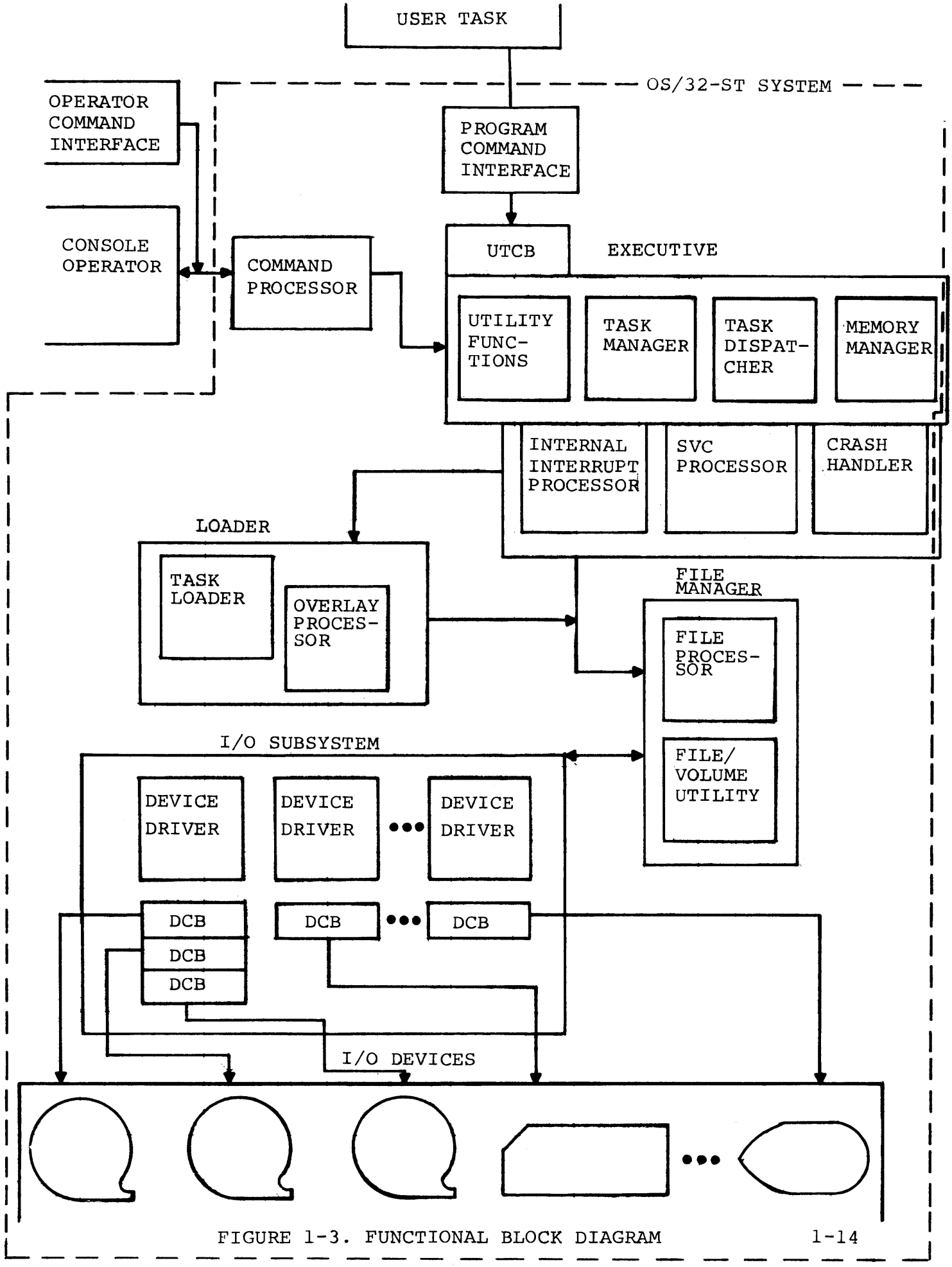


FIGURE 1-3. FUNCTIONAL BLOCK DIAGRAM

user tasks request services of the system by executing Supervisor Call (SVC) instructions (see Chapter 3). Many operator command features are made available to the user program via SVCs.

An interface between user task and system code is provided in a system table called the user Task Control Block (UTCB). The user is not ordinarily aware of this table. It is maintained by the system and contains task-related items such as register save areas, the task's Options, and the task's current Status. An important item in the UTCB that should be explained here is the task's Logical Unit Table (LTAB) which allows I/O operations to be device independent. Programs simply refer to Logical Unit (LU) numbers instead of to specific devices. Each LU number corresponds to a slot in LTAB, ranging from zero to a SYSGENable limit up to 254. The LUs used by a task are assigned to devices or files as appropriate for each given task. Use of an unassigned LU is considered illegal; the user should specify the NULL device if an assignment is required but no operation is to be performed.

1.5.1 Executive

The executive is the heart of the OS/32-ST system. It contains logic for processing SVCs and other internal interrupts, a memory manager, a task manager, a dispatcher, a crash handler, and general utility routines.

The memory manager is responsible for memory allocation as illustrated in Figure 1-2.

The task manager and dispatcher are responsible for controlling the state of the current task, saving a task's environment (registers, PSW, etc.), and restoring it.

The crash handler is entered on the occurrence of catastrophic events such as memory parity error.

Utility routines perform format conversion, mnemonic scan and other commonly used functions.

1.5.2 Command Processor

The command processor accepts command strings from the system console device, decodes them, and determines what action should be taken. It provides the interface between the operator and the system. It is SYSGENable such that the execution routines for unneeded commands may be removed from a given system configuration.

1.5.3 Loader

The OS/32-ST resident loader loads tasks and overlays. The input medium must be in one of two loader formats: The CAL 32-Bit object output, for fullword mode tasks, and the CAL 16-Bit (or OS Assembler or FORTRAN IV) object format for halfword mode tasks. Since the two modes have disparate formats, two loaders are provided in a full OS/32-ST system: the halfword mode loader and the fullword mode loader. The halfword mode loader may be SYSGENed out.

The resident loader in OS/32-ST does not link object modules. An assembled program could be loaded and executed as a task, but if that program requires other modules to be linked with it as one composite program, the modules must be either linked with the OS/32 Library Loader (Program Number 03-065) prior to loading under OS/32-ST or reassembled as one complete program. The linking of a data Common module or overlays requires the use of the Library Loader.

1.5.4 I/O Subsystem

The I/O subsystem is composed of peripheral device drivers, System Queue Service handler, I/O coordination routines, Device Control BLocks (DCBs), and other system tables. Drivers are system routines that control the actual I/O transfers. They are activated and terminated by the executive. Initiation and termination phases of drivers are interruptable and execute as though they were reentrant subroutines of the user task.

Multiple devices of the same type may be controlled by one common driver, but each device requires a separate DCB. Drivers are autonomous and may transfer data concurrently with an executing task in a manner transparent to the user.

The System Queue is a built-in feature of the Processor that causes an internal interrupt when an entry is in the queue and the Processor executes a Load PSW instruction. This facility is a convenient means for system routines to queue system events. Drivers use it, for example, on termination to

interrupt the user task in an orderly manner (transparent to the user) and to envoke the Executive.

1.5.5 File Management

File management routines allow direct access files to be created, named, protected, examined, and deleted. These routines may be SYSGENed out. Files are described in the following section.

1.6 DIRECT ACCESS FILES

1.6.1 Volume Organization

Each direct access volume contains a volume descriptor occupying Sector Ø, Cylinder Ø on the volume, in which information required by the system is maintained. This descriptor consists of five fields.

FIELD 1	FIELD 2	FIELD 3	FIELD 4	FIELD 5
VOLUME NAME (4 CHARACTERS)	POINTER TO FILE DIRECTORY	POINTER TO OS IMAGE	SIZE OF OS IMAGE	POINTER TO ALLOCATION MAP

Field 1 contains the name by which the volume is known to the system.

Field 2 contains a pointer to the file directory located on the volume.

Field 3 points to an image of the OS suitable for Bootstrap Loading. This image is placed on the volume by an operator command (INITIALIZE).

Field 4 contains the size of the OS Image in sectors.

Field 5 points to an allocation map maintained by the system. Allocation of direct access space is by 256 byte sectors. This map records unused and allocated sectors on the volume. It occupies one sector for every 500,000 bytes on the volume.

The file directory contains information needed by the system to process files recorded on the volume. An entry in the directory is made for each file.

The directory itself is organized as a chained file of one-sector blocks. A Directory Block has room for up to five file entries.

When a direct-access volume is cleared by an operator command, (INITIALIZE), the file directory pointer in the Volume Descriptor is set to zero indicating that no Directory Blocks are present. When the first file on the volume is allocated, a Directory Block is also allocated and its address is placed in the Volume Descriptor. The first file entry in the directory is then marked as in use and the remaining four as not in use. Four more files may now be allocated before a new Directory Block is required. As new Directory Blocks are added, they are chained to the end of the file directory.

1.6.2 Identification of Files

An OS/32 file is identified by name. The full name of a file has three parts: volume name, file name, and extension.

The volume name is composed of from one to four alphanumeric characters, of which the first character must be alphabetic. This is the name of the volume on which the file resides.

The file name consists of from one to eight alphanumeric characters, of which the first must be alphabetic. This is the main identifier for the file, and may be anything the user chooses.

The extension consists of up to three alphanumeric characters. It may consist of no characters at all, in which case it is considered to be made of blanks. The extension usually denotes the type of material on the file. It may be anything the user chooses; however, some specific extensions are used by OS/32 and by some OS utilities, and are assumed to have specific meanings. These extensions are:

OBJ	Absolute or Relocatable loader format.
FTN	FORTRAN source format.
CAL	CAL assembly language source format.
LST	Printer-image listing format.
BAS	BASIC source format.
CSS	OS/32 ST Command Substitution System (CSS) source format.

The user may use any of these standard extensions, or may define his own.

File identifiers, called File Descriptors in this manual, are written as follows:

```
VOLN:FILENAME.EXT
```

where VOLN is the volume name, FILENAME is the file name, and

EXT is the extension. VOLN and EXT may be omitted when default names are assumed, such as system volume and blank extension.

A File Descriptor may also be used to describe a non-direct access device, in which case the VOLN field describes a device name rather than a volume name. The colon following the device name must be retained to avoid confusion with a file specification having a default volume name and extension. The FILENAME and EXT fields are ignored for non-direct access devices and should be omitted. At SYSGEN time each device in the system is assigned a permanent four-character name by which it is called. Thus the physical address of each device need only be known at SYSGEN time. From there on, all references to each device are by name.

Throughout this manual the term File Descriptor is used to mean either a direct access file or a device. Examples of File Descriptors are given in Chapter 2.

1.6.3 File Access Methods

OS/32 supports two methods of access to files: random and sequential. These methods may be intermixed without having to close and reopen the file. The chief mechanism used to implement these methods is the current record pointer.

The current record pointer is a number, ranging from zero to the number of records currently in the file, indicating the

record to be read or written on the next sequential access to the file. Each record is numbered in sequence, starting with zero.

The current record pointer may be adjusted in one of several ways:

1. It is set to zero by the following operations:

   Rewind

   Backspace to filemark (except on Contiguous files)

   Assigning (except for write-only access)

2. It is set to the number of records in the file (the proper position to append new records) by the following operations:

   Assigning for write-only access

   Forward to filemark (except on Contiguous files)

3. It is decremented by one by a backspace record operation, unless the file is already positioned at its beginning.

4. It is incremented by one as follows:

   Forward record (unless already at end of file)

   Sequential read or write

5. A random read or write sets the current record pointer to a value one greater than the record read or written.

Random Access

For random access, the user supplies the record number that he wants to read or write. This record is found, the data

transfer is performed, and the current record pointer is set to point to the next sequential record. If the user continues to use random access, he need not pay attention to the current record pointer, since it is readjusted on every call. However, the user may wish to read or write a sequence of records, starting with a known record number. In this case, he would use a single random call and follow it with a number of sequential calls.

With a Chained file, the user is somewhat restricted in his use of the random write call. He may use this call to update any record currently in the file, or to append one record to the end of the file. If the record number specified is more than one record past the end of the file, the call is rejected with EOM (end of medium) status. Effectively, this means that a file must be expanded in a sequential manner. If the file has only five records, a sixth may be added but record number 100, for example, could not.

On Contiguous Files there is no restriction on the use of the random write or read call. Any record within the file's allocation may be read or written.

Sequential Access

Sequential access is probably the simplest and most common access method. The user performs a series of sequential read or write calls. These cause records of the file to be read or written in sequence. The current record pointer is adjusted

automatically at each access. The Rewind, Forward Record, Backward Record, Forward File and Backward File commands may be used for repositioning as described above.

1.6.4 File Organization

A file is a collection of related records. From a programmer's point of view, a file is made up of logical records which may be of arbitrary length and structure, and are process-dependent. From the system's point of view, a file is made up of physical records which are of fixed length appropriate to the particular device and are process-independent. When a user program is written, the logical file structure must be considered because certain information is required at execution time by the I/O processor routines and therefore must be supplied by the user program. When a file is allocated, the manner in which the data is to be stored physically on the device must be specified.

OS/32-ST supports two file structures. Although these structures differ, in many cases the same data manipulations can be performed on one structure as another. The choice of file structure, in most applications, does not depend on the form of the data to be put in the file, but on the way in which the data is accessed. OS/32-ST file structures are each optimized for one specific form of access.

Chained Files

The Chained file is an open-ended file structure consisting of a chain of blocks. One fullword of each block is used by the system as a pointer (this pointer is not available to the user).

The pointer field of each block points to the next block in the chain. By following the pointers, all the blocks in the file can be found, no matter where they are scattered on the volume. The pointer is bidirectional; that is, it can be used to follow the chain backwards as well as forwards. The chain in anchored at each end in the file directory. The chained structure of this file is completely transparent to the user.

When a Chained file is created (allocated), the physical block size of the file may be specified in multiples of a disc sector (256 bytes). The user's logical record size may be independent of the physical block size, however. Blocking and deblocking of logical records is performed automatically by the system via two FCB buffers allocated by the system when the file is assigned. The size of each buffer is the physical block size of the file.

Because of its chained structure, the Chained file is optimized for sequential access. In order to proceed from any one block to any other, all intervening blocks must be read. This is the normal case in sequential access, but is a waste of time in random access, unless the distances between successive random accesses are small.

Proceed I/O is not supported on Chained files; the user must wait for the transfer to complete.

Shared write access (see Section 1.6.5) is not permitted on Chained files. (If shared write access is requested, the system automatically grants exclusive write access if possible).

Contiguous Files

The Contiguous file is a fixed-length file structure. All blocks of a Contiguous file are allocated contiguously on the volume. The file size in 256 byte sectors is specified at the time of allocation, and all required space is actually reserved at that time. Each sector is considered a record to the system.

Random reads and writes may access any record on the file, regardless of which records have been previously written. This makes it possible to create a Contiguous file in a truly random fashion.

Contiguous file I/O is non-buffered and transfers a variable amount of data directly to or from the task's buffer. All transfers begin on a sector boundary and end whenever the number of bytes specified have been transferred. Although the user may transfer data in logical record blocks of different size than sectors, he must specify the appropriate sector number to position the file for random access.

The Contiguous file supports a pseudo filemark capability that gives it some of the characteristics of a magnetic tape device. The filemark is X'1313' at the beginning of a record or block. Care should be taken to ensure that this datam is not inadvertently written at the beginning of a record if filemark operations

are going to be used. The forward-file and backward-file operations on a Contiguous file function as they would on a magnetic tape. That is, the file is positioned forward or backward respectively until a filemark (X'1313') is found. The current record pointer is then left following this filemark. The write-filemark operation results in writing X'1313' at the beginning of the current record. Note that a forward-file or backward-file operation positions a Contiguous file at the end or beginning, respectively, if no filemark is found, but End-of-Medium error status results.

1.6.5 File and Device Protection

Files and devices may be protected in one of two ways: statically or dynamically.

Static Protection

Each file or device has associated with it two protection keys, one for read access and one for write access. Each key is one byte long and may have any value from X'00' to X'FF'. The values X'00' and X'FF' have special meanings. If the values of the keys are within the range X'01' to X'FE', the file or device may not be assigned for read or write access unless the operator or requesting task matches the appropriate keys.

If a key has a value of X'00', the file or device is unprotected for that access mode. Any key supplied is accepted as valid.

If a key has a value of X'FF', the file is unconditionally protected for that access mode. It may not be assigned for

that access mode to any task, regardless of the key supplied.

Some examples of static protection follow:

Write Key	Read Key		Meaning
00	00	....	Completely unprotected.
FF	FF	....	Unconditionally protected.
07	00	....	Unprotected for read, conditionally protected for write (user must supply write key = X'07').
FF	A7	....	Unconditionally protected for write, conditionally protected for read.
00	FF	....	Unprotected for write, unconditionally protected for read.
27	32	....	Conditionally protected for both read and write.

The protection keys of a file are defined when the file is allocated, and may be changed by the console operator or by any task having that file assigned for exclusive access. If the task has the file assigned for exclusive read it may change the read protection key; if it is assigned for exclusive write, the write key may be changed; and if the task has the file open for ERW, it may change either or both keys.

The protection keys of a device are defined at SYSGEN time and may be changed by the console operator only.

Dynamic Protection

When a file or device is assigned to a task, the user may wish to prevent console operations from accessing that file or device while it is being used. For this reason, the user may ask for exclusive access privileges, either for read or write, at assignment time. This form of protection is termed dynamic because it is only in effect while the file or device is actually assigned.

The access privileges are generally known by their abbreviations. These are:

SRO	Sharable read-only.
ERO	Exclusive read-only.
SWO	Sharable write-only.
EWO	Exclusive write-only.
SRW	Sharable read-write.
SREW	Sharable read, exclusive write.
ERSW	Exclusive read, sharable write.
ERW	Exclusive read-write.

A task may change its access privileges on a file without having to close the file. This is only possible if the proper conditions are met. For example, a task having a file assigned for shared read may not change it to exclusive read if the file is also assigned for shared read on another Logical Unit. Access may always be changed from exclusive to shared, however.

If the user attempts to change his access privileges, and for some reason is unable to get the new privileges, the old access privileges remain.

CHAPTER 2

OPERATOR'S GUIDE

2.1 SYSTEM START-UP

The initial system to be loaded is either the INTERDATA supplied OS/32-ST STARTER program (Program Number 03-075), or an OS/32-ST system SYSGENed to user specifications. The procedure is the same for either system.

To load the initial OS/32-ST or to reload from any device other than disc requires the use of the 32-Bit Relocating Loader (Program Number 03-067). If the system is equipped with a disc, the operator may, when initializing the disc, copy a memory image of the system onto the disc. The system may then be reloaded from disc with the OS/32 Bootstrap Loader (Program Number 03-074).

The OS/32-ST STARTER program is described in the OS/32-ST Program Configuration Manual, Publication Number 29-379. Use of STARTER may require manually changing the peripheral device addresses assigned in the program if different from those in the user's configuration. These modifications are also explained in the OS/32-ST Program Configuration Manual, Publication Number 29-379.

OS/32-ST is started (and restarted) at location X' '. During system initialization, OS/32-ST attempts to mark all SYSGENed devices on line. If a direct access is SYSGENed, but not physically present or it is not ready, a message is issued and it is marked OFFLINE. Initialization also establishes the default Logical Unit (LU) assignments and task options, resets task memory allocation, and clears the display panel.

After initializing itself, OS/32-ST outputs the asterisk character, (*) to the system console and the operator may then enter commands.

2.2 SYSTEM ERRORS

There are three kinds of error conditions that can occur during operation of OS/32-ST: system crashes, task crashes and recoverable errors.

A system crash occurs when a hardware or software malfunction is detected during execution of system code and the system cannot proceed without running the risk of destroying information, either on some peripheral or in memory. At which point, the system attempts to display a crash code on the display panel and write the crash code in dedicated memory. It then stops. The system crash code meanings and actions to be taken for each code are listed in the OS/32-ST Program Logic Manual, Publication Number 29-381.

System error conditions caused by machine malfunction interrupts occur for memory parity error, primary power failure, and power restoration.

A memory parity error causes a system crash if it occurs in the system or a task crash (the task is terminated) if it occurs in the task.

If primary power failure occurs, the registers are saved and the system prepares itself for another interrupt upon power restoration. When the machine malfunction interrupt occurs upon power restoration, a message is logged showing the occurrence of the power failure and to request the operator

to restore any devices that may be in an off-line and/or write-protected state. The operator may then continue the power restoration process by entering the command GO. At this point, all non-direct access I/O is aborted, direct access I/O is retried where necessary, and a message is logged to indicate that power restoration is complete. An interrupted active task is then PAUSEd.

A task crash occurs when a malfunction is detected during the execution of a user task and the system cannot continue executing the task without destroying the system or user information. At this point, the system prints an error message which describes the error detected (e.g., Illegal Instruction, Illegal SVC, Arithmetic Fault) and the location of the instruction causing the error. The user task is then PAUSEd. The operator may then correct the error condition and use the CONTINUE command to proceed; or the operator may abort the task with a CANCEL command.

CAUTION:

Any PAUSEd task with active I/O Proceed calls may have its I/O still on-going. An I/O Wait call is completed first, however.

A recoverable error occurs when the system detects a condition with incomplete information. In this case, the system prints a message describing the condition (e.g., Command syntax error, device unavailable, illegal command). The operator may then issue operator commands to correct the error.

2.3 SYSTEM CONSOLE DEVICE

The system console device is normally considered to be an ASR Model 33 or 35 Teletypewriter (TTY), but it may be any device (e.g., keyboard/CRT) that is TTY program-compatible. It has a special relationship with the system; operator command input is read by the Command Processor only from the console device and messages may be logged to the console device without reference to device name or address. Under certain conditions, commands may be read from another device, but this process must be started by an operator command entered at the system console.

When data must be entered from the system console in order for OS/32-ST or a user task to proceed, a character is output to the console to prompt the operator. This character is an asterisk (*) if command entry is required, or a right angle-bracket (>) if data for the user task must be entered.

When a user task is executing,certain commands may be entered from the system console; however, as the task is capable of proceeding without further command input, no asterisk is typed out. The asterisk appears whenever the task goes to End-of-Task.

If a task is in the process of reading from or writing to the system console, and the operator wishes to input a command, he should depress the BREAK key on the console device. This forces the system into command mode for the entry of one command line. After a command line has been accepted, the user's I/O to the console is restarted (unless the command cancelled the user task).

The BREAK Key may also be employed to stop and discard any response message to a command.

The user-data request prompt (>) is only output to the console device; it is not output on user read requests to any other device in the system.

When entering commands at the system console, the operator may make corrections to his input line. A back arrow character (←) causes the previous character to be ignored; multiple back arrows cause the deletion of a corresponding number of consecutive previous characters, up to, but not exceeding the beginning of the command line. A hash mark character (#) causes the entire command line to be ignored and a new asterisk prompt character to be output.

2.4 OPERATOR COMMANDS

A general definition of the command syntax follows; expressions for the formal rules are given in Appendix 1.

2.4.1 Command Syntax

Commands are accepted one line at a time, where multiple commands may appear on one line separated by semi-colons (;). The command line is terminated by a carriage return character unless the input data buffer is filled first (the size of this buffer is a SYSGEN parameter).

If the first character of any command is an asterisk (*), the remainder of the command line is considered to be a remark and is ignored, although it is copied if commands are being logged, as explained in Section 2.4.3.

Commands are composed of:

- Mnemonics
- Decimal numbers
- Hexadecimal numbers
- Arbitrary character strings
- File Descriptors

A mnemonic is shown in this manual in upper-case letters. Each mnemonic requires a minimum number of characters to be entered by the operator. These required characters are underlined, e.g.,

COMMAND

Further characters, if entered, must conform to the correct sequence of the characters of the full command word. For example, given the above COMMAND:

```
COM
COMM
COMMA      these are all legal forms of the
COMMAN     command shown above
COMMAND

CO         illegal, too short
COMX       illegal, unrecognized character
COMMANDZ   illegal, too long
```

Mnemonics consist of alphabetic characters, except for the first character which may be any ASCII character. A mnemonic is terminated by the occurrence of any non-alphabetic character (except for the first).

When a decimal or hexadecimal number is required in a command, leading zeros may be omitted.

An arbitrary character string field in a command, consisting of ASCII characters, must be the last field in a command line.

A File Descriptor (illustrated as fd in command formats) is defined as follows:

```
VOLN:FILENAME.EXT
```

Where VOLN is the volume name, FILENAME is the file name, and EXT is the file name extension. The volume name need not

be specified; the default is the system volume. The extension need not be specified; the default is usually the blank extension, although some commands may make use of a different default value. If the extension is not entered, the period following FILENAME need not be entered.

The VOLN field may consist of up to 4 characters, FILENAME may consist of up to 8 characters, and the maximum length of the EXT field is 3 characters. Each name field must consist of alphanumeric characters and the first character of the VOLN and FILENAME fields must be alphabetic only.

File Descriptors may refer to non-direct access peripheral devices as well. In this case, VOLN is the four-character device mnemonic and FILENAME and EXT should not be entered. The colon following the device mnemonic must always be entered. Examples of legal File Descriptors are:

FRED:PROGRM.TSK	
PROGRM.TSK	The same file, but on the system volume.
ABLE:PROGR3	Default extension value.
PROG64	Default volume name and extension fields.
CARD:	Name of card reader device.

Some commands have optional operands. These are annotated with brackets surrounding the entire optional part of the command, including punctuation, e.g.,:

COMMAND xxxx, (yyyy) (,zzzz)

Note that the comma preceding the operand yyyy is not optional, but the comma preceding the operand zzzz is optional.

Operands illustrated in lower-case characters mean that they represent a variable item that the operator must enter. Upper-case operands imply mnemonic fields and must be entered as illustrated. For example,

in BIAS adrs

The operand adrs must be specified according to the value (address) the operator wants.

in DISPLAY LU

the character L (or characters LU) must be entered as specified.

The repertoire of operator commands is summarized in Appendix 1. The command descriptions follow in this chapter. Examples of how commands may be used are given in Chapter 4.

If, for some reason, command input is not acceptable to the system, an error message is issued to the system console device:

XXXX-ERR

where XXXX is an alphanumeric error code of four or fewer characters. These error messages and other system messages are listed in Appendix 1. They are also mentioned with each command description as appropriate.

The error response to an unrecognized command is:

MNEM-ERR

Any commands following an unrecognized command on a command line are ignored. If entering commands while a task is active,

the message

SEQ-ERR

is issued if the command cannot be accepted while the task is active.

2.4.2 Task Related Commands

OPTIONS

The Options command specifies certain options related to the program to be loaded or started. The syntax of the Options command is:

OPTIONS opt[,opt]...

where the operands, opt, may be any one of the following:

HALF	Specifies halfword mode. This is valid only if the system is SYSGENed to support a halfword mode program.
FULL	Specifies fullword mode.
ET	Specifies that the program is an executive task (i.e., privileged task; this option should only be used with considerable caution).
UT	Specifies that the program is a User task (i.e., unprivileged) task
AFCONT	Specifies that the program is continued, with a message logged, if an Arithmetic Fault is detected.
AFPAUSE	Specifies that the program is PAUSEd if an Arithmetic Fault is detected.

Note that the options are paired (i.e., HALF/FULL, ET/UT, AFCONT/AFPAUSE), but any option may be entered and only the last one for a pair is accepted as the correct option. For example, in:

```
OPT H,F
```

fullword mode is specified.

The OPTIONS command is necessary only if the operator wishes to change the default options defined at SYSGEN time. The RESET command, as explained later, restores the default options. This command is rejected when a task is active.

Possible response messages to OPTIONS are:

*	Command accepted; another command may be entered.
SEQ-ERR	Command entered while task active.
FORM-ERR	Command syntax error.
PARM-ERR	Operand syntax error
NOPR-ERR	No operand entered.

LOAD

This command causes an object program to be loaded from a device or file. The program is loaded starting at the current value of UBOT (bottom of user program space - see Memory Management Section 1.4), unless a bias value is specified. The command syntax is:

LOAD fd [,i [,p]]

where i is the optional Impure bias of the program to be loaded and p is its optional Pure bias. Both i and p must be hexadecimal numbers of 6 digits or less. For a direct access file, the default extension for the file descriptor is .OBJ.

A program's Pure and Impure segments are defined by CAL assemblies. No indication is given if the program does not contain a segment corresponding to the specified options. The new UTOP is located above either the Pure or Impure segment, whichever is greater.

Possible response messages to LOAD are:

*	Command accepted
FORM-ERR	Command syntax error
PARM-ERR	Operand syntax error
NOPR-ERR	No fd entered
LDR1-ERR	Loader I/O error
LDR2-ERR	Memory size error
LDR3-ERR	Invalid load format item
LDR4-ERR	Insufficient space between specified pure and impure biases
LDR5-ERR	No Halfword mode support

EXPAND

This command expands a task's memory allocation by increasing CTOP (see Section 1.4 on Memory Management). The task's allocation may be expanded in increments of 256 bytes up to the top of available memory. Note, however, that too great an allocation for the task prevents the opening of direct-access files because the Operating System uses the area between the top of the program's allocation and the top of memory for direct-access File Control Blocks (FCBs) and buffers. Command format is:

EXPAND n

where operand ,n, is the number of 256 byte segments for expansion, in decimal.

Response messages to EXPAND are:

*	Command accepted
FORM-ERR	Command syntax error
PARM-ERR	Invalid operand
NOPR-ERR	No operand entered
MEM-ERR	Insufficient memory

START

This command may be used to begin execution of any program in memory. Once started, the executing program and all the routines it could execute constitute the currently active task. Command syntax is:

START [adrs][,args to prog]

where adrs represents the address at which execution is to be started. If adrs is not specified, the most recent program loaded is started at the transfer address specified when assembled. The START command is rejected if a transfer address does not exist and a start address is not given.

The optional field, args to prog, contains arguments that are to be passed to the task for its own decoding and processing. All characters between the comma beginning the field and the next terminator (semi-colon or carriage return) are moved to memory beginning at UTOP, except that any characters that would be located above CTOP are ignored.

The START command must be the last command, other than a comment, on the command line. Any other commands appearing on the command line after START are ignored.

NOTE

Once a task is started, the asterisk prompt is not output to the system console until the task stops executing and the system is waiting for a new command. The system does, however, accept all commands except the following commands while the task is executing:

LOAD
START
CONTINUE
RESET
OPTIONS
CSS call

} not acceptable while task is active

Possible error responses to START are:

SEQ-ERR	Command entered while task active
FORM-ERR	Command syntax error
PARM-ERR	Operand syntax error
NOPR-ERR	No start address given
SLOC-ERR	Invalid start address given

PAUSE

Entering this command halts the running program and reenters OS/32-ST. The program is stopped as though it had executed a PAUSE SVC and consequently may be CONTINUEd at the next instruction. Command format:

PAUSE

Any on-going Proceed I/O is allowed to continue to its normal completion after the PAUSE, but Wait I/O is completed first. Note that the START command would not continue execution at the next instruction unless so specified. The PAUSE command is rejected if the task is not active at the time PAUSE is entered.

Possible response messages to PAUSE are:

TASK PAUSED	
*	Command accepted
SEQ-ERR	Task not active
FORM-ERR	Command syntax error

CONTINUE

This command causes a task which has executed a Pause SVC or has been stopped by the operator to resume operation where it was Paused. Command format:

CONTINUE

As with the START command, the asterisk prompt is not output until the task stops executing.

Possible response messages to CONTINUE are:

*	Command accepted
SEQ-ERR	Command entered while task active
FORM-ERR	Command syntax error

CANCEL

The CANCEL command terminates a program as if it had gone to End-of-Task (EOT). Any I/O in progress is terminated and all assigned Chained files are Checkpointed (see Chapter 3). Any LU assignments remain in effect. The format of this command is:

CANCEL

to which the system responds with an EOT message if successful. No further commands should be entered until the EOT message is output.

Possible response messages to CANCEL are:

END OF TASK,255	
*	Command accepted, return code = 255
SEQ-ERR	Task not active
FORM-ERR	Command syntax error

2.4.3 General System Commands

DISPLAY

This command causes certain system and/or task information to be output to the system console or to some other device or file. Several options of the DISPLAY command exist. In each, the optional File Descriptor (fd) argument indicates the device or file to which the information is output. If fd is omitted, the system console device is assumed. Allowable options and the data displayed are as follows.

a) Logical Unit option:

DISPLAY LU[,fd]

displays the following information for assigned LUs:

LU fd

b) Device option:

DISPLAY DEVICES[,fd]

displays the following information for each device in the system:

Device Name Physical Address On/Off Line Keys Access Priv.

c) Files option:

DISPLAY FILES,voln:{filename / or}.{ext / or}[,fd]

displays directory information about file(s) currently resident on the direct access volume specified by voln. The volume name field, voln:, must be specified even for the system volume. The filename and extension (ext) fields must contain either a name or a dash (-) character. The '-' character is interpreted to mean all.

For example,

DISPLAY FILES,ABC:-.-

displays all files on volume ABC.

DI F,ABC:FILEN.-

displays all files on volume ABC with name FILEN and any extension name.

DI F,ABC:-.CAL

displays all files on volume ABC with extension CAL.

DI F,ABC:MYFILE.OLD

displays information about MYFILE.OLD only.

The information displayed is:

File Name Type Size/Logical Rec. Length Keys Access Priv.

d) System Parameters Option:

DISPLAY PARAMETERS [,fd]

displays system parameters related to the task:

- MTOP
- FBOT
- CTOP
- UTOP
- UBOT
- Transfer address
- Pause address
- Number of LUs
- Options

Possible error responses to DISPLAY are:

FORM-ERR	Command syntax error
PARM-ERR	Operand syntax error
NOPR-ERR	Operand(s) missing
SVC1-ERR	I/O error
NODA-ERR	No direct access support

SET LOG COMMAND

The SET LOG command gives the operator the ability to produce a copy of all system console I/O. This copy includes the following:

- All command lines entered from the console or Command Substitution System
- All responses to these commands
- All messages logged by the user task

The syntax of this command is:

SET LOG[fd [,COPY]]

The copy is produced on a file or device specified by the fd operand. This device may be changed at any time by another SET LOG command. If no operands are specified, logging is terminated. Logging is automatically terminated under the following conditions:

I/O error on the log device

System initialization

Power restoration

Console interrupt from the display panel

The SET LOG command may be used for two primary purposes. These are:

to provide a historical record of system operation, often on a magnetic tape or direct access file.

To allow system output, e.g., displays, maps, log messages, etc. to proceed on a high-speed device rather than on the system console.

If the optional COPY operand is specified, the system console receives all outputs that it would have received if no SET LOG command were in effect. This facility would normally be used when the logging is basically for historical purposes. If COPY is not specified, however, the system console receives no outputs other than error responses to commands entered from the system console. This is the case when logging is directed at a high-speed printer device used by the console operator as an adjunct to the system console.

The Log device may be shared with user task output.

Possible error responses to SET LOG are:

FORM-ERR	Command syntax error
PARM-ERR	operand syntax error
FD-ERR	Invalid fd
ASGN-ERR	Log device could not be assigned

BIAS

The command:

BIAS [adrs]

is used to establish a basis for the EXAMINE and MODIFY commands.

The operand, adrs, is a hexadecimal bias, to be added to any address given in a subsequent EXAMINE or MODIFY command. If adrs is omitted, all addresses specified in subsequent EXAMINE and MODIFY commands are assumed to be absolute physical addresses. Subsequent BIAS commands override all previous BIASes.

BIAS is not reset on initialization or via the RESET command; the operator should enter a new BIAS if he is unsure of its current value.

Possible error responses to BIAS are:

FORM-ERR	Command syntax error
PARM-ERR	Invalid address syntax

EXAMINE

The EXAMINE command is used to examine the contents of memory.

EXAMINE adrs[,n]

causes the contents of the memory location specified by adrs (modified by any previous BIAS command) to be displayed. The display address is rounded down to the nearest halfword. If the decimal operand n is specified, it indicates the number of halfwords to be displayed beginning at adrs. If n is omitted, a value of 1 is assumed.

Possible error responses to EXAMINE are:

FORM-ERR	Command syntax error
PARM-ERR	Operand syntax error
NOPR-ERR	Address not entered

MODIFY

The MODIFY command is used to change the contents of memory.

MODIFY adrs [,data] [,data] ...

causes the contents of the location specified by adrs (modified by any previous BIAS command) to be replaced with data. The modify address is rounded down to the nearest halfword.

If the operand, data, is omitted, the modify address has its contents replaced with zeroes. Each data field consists of 0-4 hexadecimal digits which, representing a halfword, is to be put into memory starting at the location specified by adrs. Any string of data less than four characters is right-justified and left-zero filled.

Possible error responses to MODIFY are:

FORM-ERR	Command syntax error
PARM-ERR	Operand syntax error
NOPR-ERR	Address not entered

RESET

This command causes system variables to be reset to their SYSGEN-established values. In particular, the LU table is reset to the default values established at SYSGEN time, user program space is released to UBOT, and default options (see the OPTIONS command) are restored. Command Format:

RESET

This command is rejected if entered while a task is either executing or in a PAUSEd state.

Possible error responses to RESET are:

SEQ-ERR	Command entered while task active
FORM-ERR	Command syntax error

VOLUME

This command specifies the name of the system volume. This specification overrides all specifications previously in effect. The format is:

VOLUME voln

where voln is a four-character volume identifier. The system volume name is used in any command (other than DISPLAY FILES) where the operator failed to specify a volume name explicitly.

The possible error responses to VOLUME are:

FORM-ERR	Command syntax error
PARM-ERR	Volume name syntax error
NODA-ERR	No direct access support

2.4.4 Device and File Control Commands

ALLOCATE

This command creates a file on a direct-access volume. Its syntax is:

```
ALLOCATE  fd [ { CHAINED,lrecl[/size] [,keys] }
               { CONTIGUOUS,size [,keys]      } ]
```

Where fd identifies the file to be allocated. The volume name field of fd is optional and defaults to the name of the system volume. The extension field is optional and defaults to blanks.

The fd operand is the only required operand. The remainder may be omitted as a group. If no operands follow fd, the default is as follows:

```
ALLOCATE fd,CHAINED,80/1,0000
```

This allocates a chained file whose name is specified by fd with a logical record size of 80 bytes, a physical block size of 1 sector, and read and write protection keys of zero. If this default condition is not desired, further operands must be entered. The first operand specifies the file type:

CHAINED or

CONTIGUOUS

If CHAINED is chosen, the next operand, lrecl, is required and it specifies the logical record length. The operand, lrecl, which cannot exceed 65,535, may optionally be followed by a slash mark (/) which delimits lrecl from size. The size operand (for a Chained file) specifies the physical block

size, in 256 byte sectors, to be used for buffering and de-buffering operations on this file. If size is omitted, the default value is 1 sector (256 bytes). Note that, in order to assign this file, sufficient room must exist in memory above CTOP for two buffers, each of the stated size. Therefore, if size is very great, the file may not be opened in some memory-bound situations. At SYSGEN time a certain maximum block size parameter is set up in the system. The size operand of the ALLOCATE command may not exceed this constant, which may vary from one system to another. In no case may size exceed 255. Both lrecl and size are specified as decimal numbers.

If CONTIGUOUS is chosen, the next operand, size, is required and specifies the total allocation size in 256 byte sectors. This size may be any value up to the number of contiguous sectors existing on the specified volume at the time the command is entered. This size is not to be confused with the size parameter of a CHAINED file. Size is specified as a decimal number.

If either CHAINED or CONTIGUOUS is chosen, the last operand, keys, is always optional. This operand specifies the write and read protection keys for this file. These keys are in the form of a hexadecimal halfword, the left byte of which signifies the write key and the right byte the read key. If this parameter is omitted, both keys default to zero.

Examples of the ALLOCATE command:

```
AL  THISFILE
```

Allocates on the system volume a Chained file named THISFILE. (blank extension) with a logical record length of 80 bytes, a buffer size of 1 sector, and protection keys of zero.

```
AL PROGRAM.TSK,CO,64
```

Allocates on the system volume a Contiguous file named PROGRAM.TSK, whose total length is 64 sectors (16KB) and protection keys are zero.

```
AL FRED:EXAMPLE.OBJ,CH,126
```

Allocates on the volume FRED a Chained file named EXAMPLE.OBJ, whose logical record length is 126 bytes. The buffer size of this file defaults to one sector; the protection keys default to zero.

```
AL MORT:GREATBIG.BLK,CH,132/4
```

Allocates on the volume MORT a Chained file named GREATBIG.BLK, whose logical record length is 132 bytes, using a physical block size of 4 sectors. The protection keys default to zero. Note that whenever this file is assigned, the system must have 2KB of available memory above CTOP (twice the physical block size) for buffers.

```
AL SAM:DATABASE.X,CH,480,AA55
```

Allocates on the volume SAM a Chained file named DATABASE.X, whose logical record length is 480 bytes, physical block size is 1 sector, write protection key is X'AA' and read key is X'55'. Note that the logical record length is permitted to exceed the physical block size.

Possible response messages to ALLOCATE are:

*	Command accepted, operator may verify with DISPLAY command
FORM-ERR	Command syntax error
PARM-ERR	Operand syntax error
NODA-ERR	Direct access support
ALLO-ERR	An Allocate operation failed

ASSIGN

This command assigns a device or file to a Logical Unit.

If the LU is already assigned, the command is rejected; the operator should deassign the LU via the CLOSE command.

The syntax of the ASSIGN command is:

ASSIGN lu,fd[,[access priv.][,keys]]

in which lu is the LU number in decimal, the File Descriptor (fd) signifies the file or device to be assigned, and the optional parameters specify the method of opening.

Access privileges recognized are:

SRO

ERO

SWO

EWO

SRW

SREW

ERSW

ERW

Default is SRW if omitted. The ASSIGN command is rejected if the specified access privilege cannot be granted (see Section 1.6.5 on File Protection).

The keys parameter is a hexadecimal number of which the left-hand two digits specify write protection keys and the right-hand two digits specify the read protection key. Default is X'0000' if omitted. These keys are checked against the appropriate existing keys for the file or device; the command is rejected if the keys are invalid.

An assigned direct access file is positioned at the end for access privileges EWO and SWO, otherwise the file is positioned at the beginning.

An example ASSIGN command:

```
AS 9,THISFILE,ERW
```

assigns a file, called THISFILE on the system volume with zero keys, to LU 9 with exclusive read-write access privilege.

Possible response messages to ASSIGN are:

*	Command accepted, operator may verify with DISPLAY LU command
FORM-ERR	Command syntax error
PARM-ERR	Operand syntax error
NODA-ERR	No direct access support
FD-ERR	Invalid file descripter
LU-ERR	Invalid LU number or LU assigned
PRIV-ERR	Unacceptable access privilege
ASGN-ERR	An Assign operation failed

CLOSE

This command removes a file or device assignment from a given LU. If the LU is assigned to a file, the file is closed. The command format is:

CLOSE lu

where lu is specified as a decimal number.

Possible response messages to CLOSE are:

*	Command accepted
FORM-ERR	Command syntax error
PARM-ERR	Operand syntax error
CLOS-ERR	A Close operation failed

DELETE

This command is used to delete a direct-access file. Its format is:

DELETE fd

where fd identifies the file to be deleted. To be deleted, the file must not be currently assigned to any LU. This command is not recognized if there are no direct-access devices in the system.

Possible response messages to DELETE are:

*	Command accepted
FORM-ERR	Command syntax error
PARM-ERR	Operand syntax error
NODA-ERR	No direct access support
FD-ERR	Invalid fd
ASGN-ERR	fd currently assigned
DEL-ERR	A Delete operation failed

RENAME

This command is used to change the name of an unassigned direct-access file or of a device. Its format is:

RENAME oldfd,newfd

Examples:

REN VOL1:MYFILE.CUR,MYFILE.OLD

REN MTO1:,MTO2

The volume ID field of the new File Descriptor (new fw) may be omitted for direct access files. If it is entered, the system ignores it (therefore, this command cannot be used to rename a direct access volume, USE VOLUME or INITIALIZE. The operator should not rename the console device.

Possible response messages to RENAME are:

*	Command accepted; operator may verify with DISPLAY Command
FORM-ERR	Command syntax error
PARM-ERR	Operand syntax error
ASGN-ERR	fd currently assigned
RENM-ERR	A Rename operation failed

REPROTECT

This command is used to change the protection keys of an unassigned direct-access file or of a named device. Its format is:

REPROTECT fd,keys

where the operand, keys, is a hexadecimal halfword constant specifying the new write and read protection keys.

Possible response messages to REPROTECT are:

*	Command accepted; operator may verify with the DISPLAY command
FORM-ERR	Command syntax error
PARM-ERR	Operand syntax error
ASGN-ERR	fd currently assigned
NOPR-ERR	keys not specified
REPR-ERR	A Reprotect operation failed

INITIALIZE

This command is used to initialize a direct-access volume. The driver must be in an OFF state. The system may write a clean bit-map, clear the directory, check the entire volume for bad spots, and write a new system image on the volume, depending upon the options specified. A new volume descriptor is written. Following this command, the volume may be MARKed ON. The format of the INITIALIZE command is:

INITIALIZE dm,voln[,CLEAR][,SAVE]

where dm is the four-character mnemonic name of the direct access device which contains the volume to be INITIALIZEd. Operand voln is the four-character volume identifier to be given to this volume.

CLEAR Causes the directory to be cleared, the volume checked for bad spots (which have been flagged by an off-line formatter test program) and a new bit map to be written.

SAVE Causes an image of the OS/32-ST system presently in memory to be written onto the volume. This overrides any previous OS image present on that volume. If CLEAR is not specified, and if insufficient contiguous space exists on this volume for the new image, an error message is logged, and the previous status of the disc is retained unmodified.

Examples:

```
I DIS1,SYSV,C,S
I DIS2,FRED,C
```

NOTE

If neither CLEAR nor SAVE is specified, the only action taken is to modify the volume name.

NOTE

CLEAR does not perform reformatting functions on a volume. If reformatting is required, it must be done with an appropriate test program or equivalent prior to using under OS/32-ST.

Possible response messages to INITIALIZE are:

*	Command accepted; operator may verify with DISPLAY command
FORM-ERR	Command syntax error
PARM-ERR	Operand syntax error
NODA-ERR	No direct access support
STAT-ERR	Device not off-line
SPAC-ERR	Not enough space for an OS image
PACK-ERR	Pack invalid for OS/32

MARK

The MARK command is used to take a device off-line, or place on-line a device previously in off-line mode. Its format is:

$$\underline{\text{MA}}\text{RK dm} \left\{ \begin{matrix} ,\underline{\text{ON}} \\ ,\underline{\text{OFF}} \end{matrix} \right\}$$

where the first operand (dm) specifies the mnemonic name of the device and the second operand specifies whether the device is to be MARKed as ON or OFF line. If the device is the console TTY, is assigned or has any assigned files, the MARK OFF command is rejected. Note that dm does not include a colon as required by the File Descriptor (fd) operands.

Example:

```
MA CARD,OF
```

Possible response messages to MARK are:

*	Command accepted; operator may verify with DISPLAY command
FORM-ERR	Command syntax error
PARM-ERR	Operand syntax error
ASGN-ERR	dm assigned

2.4.5 Magnetic Tape and Cassette Commands

Six commands are available to perform certain commonly required operations on magnetic tapes or cassettes. They are:

REWIND	fd	rewind
RW	fd	rewind (alias)
WFILE	fd	write filemark
FFILE	fd	forward space filemark
BFILE	fd	backspace filemark
FRECORD	fd	forward space record
BRECORD	fd	backspace record

where the File Descriptor operand, fd, is the device mnemonic name including colon (:), of the tape unit upon which the function is to be performed.

Example:

```
REWIND CAS1:     rewind cassette unit 1
```

Before executing these control commands, the Command Processor makes a temporary internal assignment of the specified device for Shared Read/Write (SRW) access privilege. The command is rejected if such assignment cannot be made (as would be the case if the device is currently assigned with exclusive access privilege). Note, however, in most applications that the default SRW access privilege would be employed for devices (see ASSIGN command above) and that this restriction would not normally be apparent to the user. Furthermore, this technique carries the advantage that a task, having assigned a tape for exclusive access, insures that inadvertent operator action cannot reposition the tape.

REWIND positions the tape at beginning of tape (BOT).

WFILE writes a filemark (defined by the hardware) and positions the tape to read/write the next record.

FFILE advances the tape to the record just past the next filemark.

BFILE retreats the tape to the previous filemark (or BOT) and then advances it to the next sequential record following the filemark.

FRECORD advances the tape to the next sequential record.

BRECORD retreats the tape one sequential record, if not at BOT.

Generally, where an analogous situation exists, these control commands may also be employed with direct access files by using a complete direct access File Descriptor instead of a device mnemonic name in the operand field. The control operations permitted with each file structure are given in Section 1.6 on Direct Access Files. The operator should understand that in order to execute these commands, the Command Processor must be able to temporarily assign the specified file. (Note, for example, Chained files require exclusive write access.) This restriction is no different than with tape devices except that the user may ordinarily use various access privileges with direct access files, and this restriction may become more apparent.

REWIND sets the record count to zero.

WFILE writes a pseudo-filemark record and increments the record count for Contiguous files; it is not supported for Chained files.

FFILE sets the record count to the number of records in the file (Position at which to append new records); the file is positioned after the next sequential filemark record on a Contiguous file.

BFILE sets the record count to zero, except for Contiguous files, in which cases the record count is set to correspond to the previous filemark record number plus one (or zero if no filemark is present).

FRECORD increments the record count by one, unless already at the end of file.

BRECORD decrements record count by one, unless the file is already positioned at its beginning.

Possible response messages to these commands are:

*	Command accepted
FORM-ERR	Command syntax error
PARM-ERR	fd syntax error
SVC1-ERR	I/O error
ASGN-ERR	Unaccessible fd/dm

2.5 COMMAND SUBSTITUTION SYSTEM

The Command Substitution System (CSS) is an extension to the OS/32-ST Command Processor. It provides the user with the ability to establish files of commands which can be called from the console and executed in a defined sequence. In this way, complex operations can be carried out by the operator with only a small number of commands. For instance, to compile, load, and run a Fortran program, only a single command need be entered.

CSS provides more than just the ability to switch the operating system command input stream to a Batch device:

A set of logical operators are provided to control the precise sequence of commands to be obeyed.

Parameters can be passed to a CSS-file so that general sequences can be written which take on specific meaning only when the parameters are substituted.

One CSS file can call another, in the manner of a sub-routine, so that the experienced user can develop complex command sequences.

Using CSS is much like writing programs, where in operator commands are strung together instead of program instructions. The following sections define the CSS 'programming' rules. Examples of CSS programs are given in Chapter 4 and in the OS/32 User Guides Manual, Publication Number 29-393.

2.5.1 Calling CSS Files

A CSS File is called and executed by naming it in a stream of commands. Any valid File Descriptor (fd) can be used, provided that there is no clash with any of the ordinary commands. If the file extension is omitted, CSS is assumed. The CSS call must be the last command on a command line. In other words, the operator can cause a file of commands to be executed simply by entering the name (fd) of the file. The error message, MNEM-ERR, is returned if the file does not exist as specified.

Parameters are passed to a CSS file by appending them to the call. The first parameter is separated from the file name by a space; all other parameters must be separated by commas. Null parameters are permitted. Leading blanks are suppressed when parameters are passed.

The following are valid CSS calls:

RUN	(Calls CSS file RUN.CSS)
CARD:	(Calls CSS file in card reader)
JUMP A,B,C	(Calls CSS file JUMP.CSS with three parameters A, B, and C)
JUMP .CSS A,B,C	(same as above)
JUMP ,,C	(Calls CSS file JUMP.CSS with three parameters, the first two of which are null)

2.5.2 Use of Parameters

Within a CSS file a parameter to that file is referenced by means of the special symbol '@'. The first parameter is referenced by @1, the second by @2, etc. A straight forward text substitution is employed.

Thus, a CSS file RUN might consist of:

```
LOAD  @1
START  @3,@2
etc.
```

This would then be called:

```
RUN  PROGRAM,NOLIST,148
```

Before each line of the CSS file is obeyed, it is preprocessed, and any reference to a parameter is substituted with the text of the parameter. Thus, the file RUN with the previous call would be obeyed as:

```
LOAD PROGRAM
START 148,NOLIST
etc.
```

In general, a reference to a parameter is of the form

```
@n
```

Where n is a decimal number indicating which parameter argument the user is referencing. Arguments are numbered starting with 1. Argument 0 is a special argument, and is defined in the following paragraph.

Being a decimal number, a reference variable is terminated by a non-decimal character. For example, to reference variable 12,

```
@12 or @12ABC or @12.EXT
```

are valid expressions.

Notice that this mechanism allows concatenation. For instance, if in the above file, RUN, the first command were

```
LOAD @1.OBJ
```

then only object files would be presented to the loaded.

Concatenation of numbers requires care. 123@1 is permitted and would expand correctly, but @1123 is a reference to parameter number 1123.

A reference to a non-existent parameter is considered to be null.

MULTIPLE @'s

CSS files can call each other to a maximum depth specified at system generation time. The multiple @ facility enables a CSS file to access parameters of higher level files. Thus, @@2 in a CSS file refers to the second parameter of the file which called this file.

For instance, given there is a CSS call,

```
CSS1 arg1,arg2
```

and suppose in file CSS1 there is another CSS call,

```
CSS2 arg3,arg4
```

then the following references may be made in CSS2:

@1 = arg3

@2 = arg4

@@1 = arg1

@@2 = arg2

If a multiple @ sequence is such that the file referred to is non-existent, then the parameter is considered to be null.

@0

This is a special case parameter. It is used to reference the CSS file in which it is contained. @0 is replaced, during the pre-process of the command line, with the name of the File Descripter in precisely the style used to call the file.

This mechanism can be used to assign the CSS file itself to a LU of a program. By this means the data for a program can be included in the CSS file itself. However, the program must read precisely the right number of data items or else subsequent CSS processing fails.

By simple extension, @@0 refers to the file which called the CSS file that contains it.

2.5.3 Commands Executable From a CSS File

Except for PAUSE, all of the commands normally available to the operator at the console can be used in a CSS file, as well as a number of commands specifically associated with the CSS facility. These additional commands are described as follows.

PAUSE is only meaningful if a program is running. CSS files are never active if a program is actually running under OS/32-ST.

Most of the CSS commands start with the character $. If a log of commands is being kept, the $'s help to emphasize where CSS has been used, but the $ has no special meaning.

$EXIT and $CLEAR

These two commands are provided for exiting from CSS files. $EXIT causes control to return to where it was when the CSS file was called. Control returns either to the console or to a higher CSS file.

$CLEAR causes unconditional return of control to the console.

$JOB and $TERMJOB

These commands delimit CSS jobs. The CSS job is a defensive mechanism which protects one user against the errors of a previous user; CSS jobs cannot be nested. Most error conditions cause the CSS processor to skip to $TERMJOB, and serious errors may return control immediately to the console.

By this means, independent users can safely mix jobs in a CSS file (e.g., a card reader), secure in the knowledge that their jobs are safe from the errors of others.

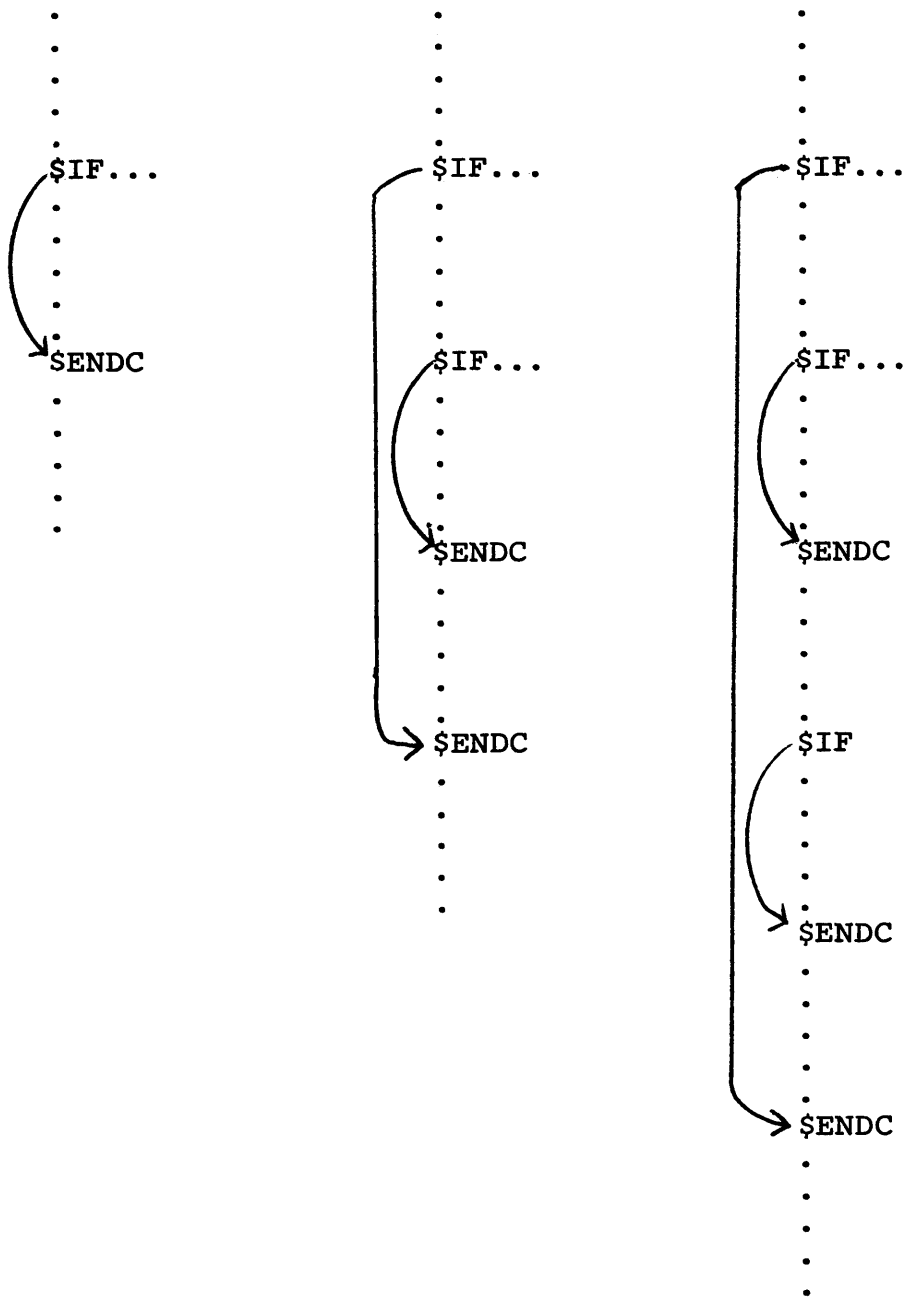
Logical Operators

There are ten logical operators available. They all start with the three characters $IF and carry one argument (e.g., $IFE 255, $IFX B.CSS, $IFNULL @1).

Each logical statement establishes a condition which is tested by the CSS processor. If the result of this test is 'true', then commands up to a corresponding $ENDC command are obeyed. If the test gives 'false' these same commands are skipped.

The $ENDC command delimits the range of a logical operator, however, nesting is permitted, so each $IF must have a corresponding $ENDC.

In the following examples, the ranges of the various conditionals are indicated by arrows.



There is no practical restriction on the depth of nesting. The logical operators fall into three categories as described below. (Return Code testing, file existence testing, and parameter existence testing).

Return Code Testing

The Return Code is a halfword quantity maintained by the system (also see description of SVC 3 in Chapter 3).

It is set in any of the following five ways:

SET CODE n - This command, which can be included in a CSS file or entered at the console, sets the Return Code to n.

$JOB - As part of its start job function, this command resets the Return Code to zero.

Command Error - Any command error causes the CSS mechanism to skip to $TERMJOB (assuming that a $JOB has been obeyed, if not, control returns to the console). To indicate that the skip has taken place, the Return Code is set to 255.

$SKIP - This command has the same effect as a command error.

EOT (SVC 3,n) - When any program terminates by executing the EOT program command (SVC 3,n) the Return Code is set to n.

There are six commands available for testing the return code:

$IFE	n	Test Return Code equal to n
$IFNE	n	Test Return Code not equal to n
$IFL	n	Test Return Code less than n
$IFNL	n	Test Return Code not less than n
$IFG	n	Test Return Code greater than n
$IFNG	n	Test Return Code not greater than n

In all cases if the test gives 'false', CSS skips commands until the corresponding $ENDC. If such skipping attempts to skip beyond a $TERMJOB or End of File, a command error is given. (see error conditions in Section 2.5.5.)

File Existence Testing

There are two commands concerned with the existence of files:

$IFX	fd	Test fd for existence
$IFNX	fd	Test fd for nonexistence

Again, if the test gives 'false', CSS skips to the corresponding $ENDC. The previous restriction on skipping also applies.

Parameter Existence Testing

There are two commands concerned with the existence of parameters:

$IFNULL	@n	Test @n null
$IFNNULL	@n	Test @n not null

Again, if the test gives 'false', CSS skips to the corresponding $ENDC, with the same restriction as previously stated.

The use of the multiple @ notation to test for the existence of higher level parameters is permitted.

In addition, a combination of parameters can be simultaneously tested. For example,

```
$IFNULL @1@2@3
```

In effect, this tests the logical AND of @1,@2 and @3 for nullity. If any of the three is present, then the test results in 'false'.

Listing Directives

Two commands are provided to control the listing of CSS files as they are executed: $COPY and $NOCOPY.

$COPY causes subsequent command lines to be listed, in their expanded form after parameter substitution. The listing takes place on the console or the log device, according to the options selected in a previous SET LOG command.

$NOCOPY switches off the listing. The default is $NOCOPY.

CSS File Construction

There are two command pairs provided for construction of CSS files: BUILD, ENDB and $BUILD, $ENDB.

The BUILD and ENDB Commands

The BUILD command causes succeeding lines to be copied to a specified file, up to but excluding the corresponding ENDB

command. The format of the BUILD command is:

BUILD fd

where fd is the new CSS file. If fd does not already exist it is created.

BUILD can be issued from the console or from within a CSS file. No nesting of BUILD commands is possible. The processing of BUILD ends when the first ENDB command is encountered, so any attempt to next BUILD commands results in a corrupt CSS file being constructed.

The BUILD command must be the last command on its input line. Any further information on the line is treated as comment and is not copied to the new CSS file.

The ENDB command must be the only command on a line, and must occupy the first four character positions on the line. Any further information on the line is treated as comment and is ignored.

A $BUILD....$ENDB sequence can be nested inside a BUILD....ENDB pair.

The BUILD....ENDB mechanism can be used to create any type of file.

The $BUILD and $ENDB Commands

These commands operate in a similar manner to BUILD and ENDB, except that before each line is copied to the CSS file, the CSS pre-processor substitutes any parameters in the line. It follows that $BUILD is only sensibly used from within a CSS file so that parameters can be passed to it. The $BUILD command has the following format:

```
$BUILD fd
```

where fd is the new CSS file. If fd does not already exist, it is created.

As with BUILD, no nesting of $BUILD is possible. A corrupt CSS file results if the attempt is made.

$BUILD must be the last command on its input line, any further information is treated as comment and ignored.

$ENDB must be the only command on its input line, and it must occupy the first five character positions on the line. Any further information is treated as comment and ignored.

A BUILD....ENDB sequence can be nested within a $BUILD....$ENDB pair.

2.5.4 CSS Command Summary

$JOB	Start next job, reset Return Code
$TERMJOB	End of job, any error skip in last job stops at this command with Return Code = 255, otherwise Return Code is defined by the job itself.

$EXIT	Exit from CSS file.
$CLEAR	Return control to console.
SET CODE n	Set Return Code to n.
$IFE n	If Return Code equals n, continue obeying commands, otherwise skip to corresponding $ENDC.
$IFNE n	If Return Code not equal to n, continue obeying commands, otherwise skip to corresponding $ENDC.
$IFL n	If Return Code less than n, continue obeying commands, otherwise skip to corresponding $ENDC.
$IFNL n	If Return Code not less than n, continue obeying commands, otherwise skip to corresponding $ENDC.
$IFG n	If Return Code greater than n, continue obeying commands, otherwise skip to corresponding $ENDC.
$IFNG n	If Return Code not greater than n, continue obeying commands, otherwise, skip to corresponding $ENDC.
$IFX fd	If fd exists, continue obeying commands, otherwis skip to corresponding $ENDC.
$IFNX fd	If fd does not exist, continue obeying commands, otherwise skip to corresponding $ENDC.
$IFNULL @n	If parameter does not exist, continue obeying commands, otherwise skip to corresponding $ENDC.
$IFNNULL @n	If parameter exists, continue obeying commands, otherwise skip to corresponding $ENDC.
$ENDC	Delimits above conditionals
$COPY	Switch on listing

$NOCOPY	Switch off listing
$BUILD	Construct CSS file with parameter substitution
$ENDB	End of $BUILD
BUILD	Construct CSS file without parameter substitution
ENDB	End of BUILD
$SKIP	Skip to $TERMJOB

2.5.5 CSS Error Conditions

ERROR	MESSAGE	ACTION TAKEN
Task active	SEQ-ERR	Control returned to console
Command not recognized	MNEM-ERR	Skips to $TERMJOB (see note)
Command syntax error	FORM-ERR PARM-ERR	Skips to $TERMJOB (see note)
Second $JOB found	JOBS-ERR	Returns control to console
End of File found while skipping to $ENDC	READ-ERR	Skips to $TERMJOB
$TERMJOB found while skipping to $ENDC within a job	READ-ERR	Sets return code to 255 and ends job. (This is only detected if the conditional that caused the skip was also inside the job; i.e., a skip to $ENDC can skip over a complete job).
End of file found before ENDB while BUILDing a file	READ-ERR	Skips to $TERMJOB (see note)

CSS Error Conditions (cont.)

ERROR	MESSAGE	ACTION TAKEN
End of file found before $ENDB while $BUILDing a file	READ-ERR	Skips to $TERMJOB (see note)
Not enough space to build an fd	FD-ERR	Skips to $TERMJOB

NOTE: 'skips to $TERMJOB' - this action only occurs if the error is detected within a CSS job. The job is abandoned and the next command obeyed is the first command after the $TERMJOB, at which point the return code is 255. If the error occurs outside a job, control is returned to the console.

CHAPTER 3

PROGRAMMER'S GUIDE

3.1 SYSTEM CONVENTIONS

Programs written to execute within the framework of OS/32-ST must adhere to certain established conventions to preserve the integrity of the system and to make the best use of I/O and other services performed by the system.

The major constraint on user programming deals with the instructions that affect Processor status. Because OS/32-ST administers all I/O and interrupts, user programs should execute in User Mode and not execute privileged instructions (instructions that relate to I/O or change the state of the Processor). Programs request I/O and other services via specific conventional calls to the system.

All user programs run with interrupts enabled, except the Arithmetic Fault interrupt may be either enabled or disabled in the Program Status Word (PSW) by a request from the user program.

On the occurrence of an Arithmetic Fault, two options are open to the user task. If the task is running with the PSW Arithmetic Fault Interrupt Enable Bit reset, the fault does not cause an interrupt and therefore is ignored. If this PSW bit is set, an interrupt occurs. A message is logged, and the program either resumes execution or PAUSEs, depending on the latest OPTIONS command (either AFCONT or AFPAUSE) entered.

NOTE

The programmer should be aware that OS/32-ST does not support memory protect.

3.2 TASK OPTIONS AND STATUS

A task may be granted certain options. Options recognized by OS/32-ST include the following:

- Executive (privileged) mode
- Continue on Arithmetic Fault
- Halfword mode

They may be set with the OPTIONS command.

During execution, a task can be in several states as reflected by its status:

- Dormant
- I/O Wait
- Task Wait (for system program)
- Console Wait

The user would not ordinarily be concerned with the task's status.

3.3 SUPERVISOR CALL (SVC)

The SVC instruction is the means whereby the user task is enabled to communicate with the Operating System and to use the facilities thereof. Execution of an SVC causes an internal interrupt which is processed by the Executive of OS/32-ST. Through the use of SVCs, a program may access peripheral devices in a rational, consistent and device-independent manner, may communicate with the system console, may modify the conditions under which it operates, and may perform a variety of useful utility functions. In short, SVCs provide the Program Command Interface to OS/32-ST.

The general form of an SVC instruction is:

```
SVC  n,N
```

where n represents the type of SVC and N represents a parameter or an address of a parameter block in the task's memory space as required by the particular type of SVC.

A parameter block may contain function codes, parameters, data fields, and status fields depending on the type of SVC. It must lie on a Fullword boundary.

SVCs recognized by OS/32-ST are:

SVC 1,PARBLK	Input/Output (I/O) operations
SVC 2,PARBLK	System utility services
SVC 3,N	End of Task (EOT)
SVC 5,PARBLK	Overlay call
SVC 7,PARBLK	File handling services

Several examples of SVCs are given in Chapter 4.

Any SVC issued by the user program that is not recognized by the system, that has a faulty or unrecognizable parameter block, or that has an address in the parameter block outside the user's memory allocation causes an error message to be issued to the system console or log device, and the program is terminated. The programmer should insure that all his buffer blocks lie on fullword boundaries.

3.4 SVC 1 - INPUT/OUTPUT OPERATIONS

All requests for I/O operations initiated by the user program are mediated by the SVC 1 Supervisor Call instruction (See Chapter 4 for example SVC 1 calls). The format of the SVC 1 instruction is:

```
SVC 1,PARBLK
```

where PARBLK is the address of a parameter block on a fullword boundary. This parameter block requires at a minimum a Function Code, a Logical Unit (LU) number and status field. Additional parameters may be required, depending on the particular command. The format of the SVC 1 parameter block is as follows:

```
            ALIGN   ADC
PARBLK      DB      X'xx'   FUNCTION CODE
            DB      X'xx'   LOGICAL UNIT
            DS      2       STATUS
            .               (additional parameters
            .               as required by the type of
            .               function being performed)
```

The Logical Unit number, which is contained in the LOGICAL UNIT byte, specifies, through the task's LU table, the peripheral device or direct-access file on which the requested data transfer or command is to take place. The LU specified in this byte must be a permissible LU number (i.e., it must not exceed the maximum LU number established at SYSTEM time), and it must have been previously ASSIGNed, either by default, via

a console ASSIGN command, or via an Assign SVC 7 call. An SVC 1 directed to an illegal or unassigned LU is returned with error status.

The Status field is used by the system in which to return the results of the SVC 1 request. This is in addition to the setting of the PSW Fullword and Halfword Condition Codes which the program can test following the SVC 1 instruction. The Condition Codes are set to X'F' if the request is rejected (see Unconditional Proceed explanation below) and the Status field is unchanged. Otherwise, the Condition Codes are set to zero and the program should ascertain the final disposition from the Status field.

The interpretation of the FUNCTION CODE byte may be divided into two classes: data transfer functions and command requests. These classes may be differentiated by the condition of Bit 0 of the FUNCTION CODE byte. If this bit is 0, a data transfer is requested; if it is 1, a command function is requested. All other bits of the function code are interpreted according to the setting of Bit 0 as illustrated in Figure 3-1.

BIT	Bit 0 = 0 Data Transfer		Bit 0 = 1 (Command)	
	0	1	0	1
1	nil	Read	nil	Rewind
2	nil	Write	nil	Backspace Record
3	ASCII	Binary	nil	Forward Space Record
4	Proceed	Wait	nil	Write File Mark
5	Sequential	Random	nil	Forward Space File
6	nil	Unconditional Proceed	nil	Backspace File
7	Formatted	Image	nil	Reserved

FIGURE 3-1. Interpretation of SVC 1 Function Codes

3.4.1 SVC 1 Data Transfer Requests

If Bit 0 of the function code is 0, a data transfer is requested. The value of Bits 1 through 7 indicates the type of transfer (Read/Write) and the modifiers selected by the program. Setting Bit 1 of the function code indicates a read operation; setting Bit 2, a write operation. If Bits 1 and 2 are both set, a Test and Set operation is requested. The modifiers offered are: ASCII/Binary, Proceed/Wait, Sequential/Random, Unconditional Proceed, and Formatted/Image. These modifiers may not all be supported on all devices.

In general, the driver accepts the logical 'OR' of all modifier bits set, and acts accordingly. Thus, the byte X'5F' is decoded as binary 0101 1111, meaning Read, Binary, Wait, Random,

Unconditional Proceed, and Image. Provided that the Binary , Random, and Image modifiers are all appropriate for the selected LU, a Read operation takes place, affected by all these modifiers.

The ASCII/Binary, Formatted/Image and Sequential/Random modifiers convey rather obvious meanings, but their specific protocol interpretations are device-dependent and are explained in the OS/32-ST Series General Purpose Driver Manual, Publication Number 29-384.

The significance of the Proceed/Wait and Unconditional Proceed options, however, is common to all devices, and is interpreted as follows:

Unconditional Proceed deals with the question of whether I/O can be started at the time of the SVC call. If Unconditional Proceed is not specified, the program waits (regardless of the setting of the Proceed/Wait bit) until the system is able to activate a driver in response to the request. If Unconditional Proceed is specified, the system permits the program to proceed if a driver could not be activated. The Halfword and Fullword Condition Codes in the PSW are set to a value of X'F' if the transfer could not be started because the device is busy. Otherwise, the condition codes are set to zero.

Once the driver has been started, the Proceed/Wait option takes on significance. If Wait is specified, the program

waits until termination of the data transfer, either normally or due to the occurrence of some error. If proceed is specified, the program is permitted to execute concurrently with the data transfer. Note that the Wait option takes precedence over Unconditional Proceed if the driver is not busy.

The Test and Set operation is currently implemented only with Contiguous files. It provides compatibility with previous INTERDATA Operating Systems. The Test and Set function allows a task to access a file or a sector within a file, and at the same time mark it as being in use. When a program issues a Test and Set, the system reads the data into the buffer, randomly or sequentially, and before returning to any user level task, checks the first halfword of the data read in. If this halfword is zero, it forces it to X'FFFF' and rewrites the complete record back in its original location. It then sets the caller's Condition Code to zero. If the first halfword is already X'FFFF', the caller's Condition Code is set to X'F'. If the first halfword is neither zero nor X'FFFF', the caller's Condition Code is zeroed. The function code for Test and Set is X'60'. Test and Set operations must specify wait (i.e., Bit 4 of the Function Code set), but is otherwise assumed. When requesting a Test and Set operation, the buffer specified in the SVC 1 Parameter Block should be a minimum of 1 sector in length.

The parameter block for an SVC 1 of the Data Transfer class is illustrated in Figure 3-2.

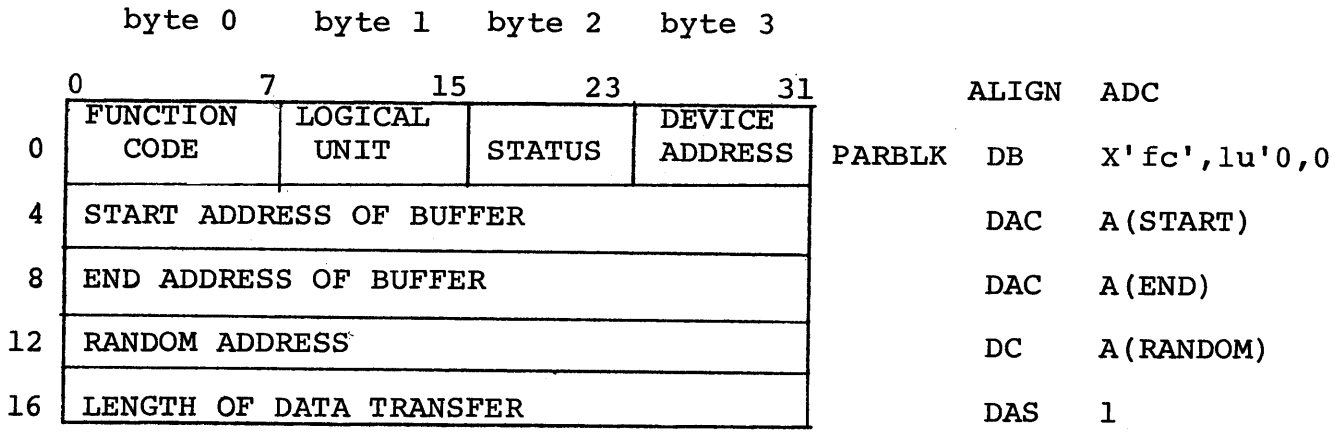


Figure 3-2. SVC Parameter Block for Data Transfers

Items in the parameter block may be redefined immediately after executing a Proceed I/O call, but the block, itself, should not be reused until the I/O completes because the STATUS and DEVICE ADDRESS bytes and the LENGTH OF DATA TRANSFER fullword are set by the system on completion of any I/O operation. The full parameter block is always required for data transfer requests, although the START ADDRESS, END ADDRESS, RANDOM ADDRESS and LENGTH fields are not necessarily used by the individual drivers in all cases. All I/O buffers must be aligned on a fullword boundary.

The LENGTH OF DATA TRANSFER field is set up by the system following any data transfer operation. It gives the length, in bytes, of the actual data transfer that took place. This field is most useful when dealing with variable-record length devices, such as magnetic tape.

NOTE

If the SVC 1 causes a Read request to be made to the system console device, a ' ' is output as a prompt to the operator.

3.4.2 SVC L Command Requests

As shown in Figure 3-1, if Bit 0 of the function code is 1, the system recognizes a command request. The implementation of all commands is device-dependent and is explained for each device in the OS/32-Series General Purpose Driver Manual, Publication Number 29-384. The implementation of commands on direct-access files is explained in Section 1.6 of this manual.

In general, the following principle applies:

The command byte is scanned from the left to right. The leftmost bit found that is meaningful to the device associated with the specified Logical Unit indicates the command to be executed.

3.4.3 Returned Status

The Operating System uses the STATUS and DEVICE ADDRESS bytes (see Figure 3-2) to return ending status to the user task upon completion of an SVC 1 request. (These fields are cleared or set, as the case may be, only if Condition Code = 0). If the termination is normal, both bytes are set to zero by the system. If the termination is abnormal, the system stores a value in the STATUS byte. In addition, it stores the physical address of the device in the DEVICE ADDRESS byte.

The DEVICE ADDRESS byte is retained for compatibility with previous INTERDATA Operating Systems; however, it should be pointed out that, in a system in which named files are commonly used, the physical device address may be singularly uninformative to the operator, and a user program should probably inform the operator of the LU on which the error occurred. This is so because the LU number is always unique within the program and the operator may easily determine the identity of the device or file that failed by means of the DISPLAY command. Alternatively, the Fetch Attributes SVC 7 call may be performed and the File Descriptor returned to the user. Moreover, the 10-bit device address is truncated to the eight least significant bits, creating the possibility of error if the device address is greater than X'FF'.

Specific interpretations of the error codes as they apply to devices are explained in the OS/32-Series General Purpose Driver Manual, Publication Number 29-384. The general definition of the status bits is given in Figure 3-3.

Bit	Meaning if set to 1	Binary	Hexadecimal
0	Always 1 for error status		
1	Illegal Function	1100 0000	X'C0'
2	Device Unavailable	1010 0000	X'A0'
3	End of Medium	1001 0000	X'90'
4	End of File	1000 1000	X'88'
5	Unrecoverable Error	1000 0100	X'84'
6	Parity or Recoverable Error	1000 0010	X'82'
7	Illegal or Unassigned LU	1000 0001	X'81'

FIGURE 3-3. Interpretation of SVC 1 Status Byte

In general, for a data request, if Illegal Function, Device Unavailable, or Illegal LU status is indicated alone, then no data was transferred. Otherwise, some data may have been transferred. Device Unavailable may indicate that the device is physically inoperative. If the device becomes unavailable after a transfer is started, Unrecoverable Error is set to indicate the possible transfer.

The Illegal Function Status bit is set whenever the system cannot accept the SVC 1 Function byte as specified. This is because the SVC 1 processor found that the requested function does not correspond to the attributes of the named device or the direct access file (for example, a read request was made to a line printer), or that the requested data transfer violates the assigned access privileges (e.g., a write request on a SRO or ERO-assigned file).

Attributes characterize a device or direct access file. The standard attributes that may be supported by a given device or file are listed in Figure 3-4.

Read
Write
ASCII/Binary
Proceed/Wait
Sequential/Random
Unconditional Proceed
Format/Image
Rewind
Backspace Record
Forward Space Record
Write File Mark
Forward Space File Mark
Backspace File Mark

Figure 3-4. SVC 1 Attributes

3.5 SVC 2 - SYSTEM UTILITY SERVICES

The SVC 2 call is used for miscellaneous service and supervisory functions. Many of the functions performed by this call are maintained for program compatibility with other INTERDATA Operating Systems. The program also uses this call for communication with the system console device via Log Message. The general format of an SVC 2 is:

```
            SVC     2,PARBLK
            ...
            ALIGN   ADC
PARBLK      DB      OPTIONS,CODE    FUNCTION CODE WITH OPTIONS
            ...                     OTHER PARAMETERS AS REQUIRED
```

The parameter block must be on a fullword boundary.

The FUNCTION CODE byte is used by the system to determine the particular type of SVC 2 call. The function codes recognized are shown in Figure 3-5.

The OPTIONS byte modifies the function code. For those functions for which no options are required, the OPTIONS byte must be set to zero.

Code Number	Meaning
1	PAUSE
2	GET STORAGE
3	RELEASE STORAGE
4	SET STATUS
5	FETCH POINTER
6	UNPACK BINARY NUMBER
7	LOG MESSAGE
8	RESERVED
9	RESERVED
10	RESERVED
11	RESERVED
12	RESERVED
13	RESERVED
14	RESERVED
15	PACK NUMERIC DATA
16	PACK FILE DESCRIPTOR
17	SCAN MNEMONIC TABLE
18	MOVE ASCII CHARACTERS
19	PEEK
20	EXPAND ALLOCATION
21	CONTRACT ALLOCATION

NOTE: For compatibility with other operating systems, Codes 8 and 9 are treated as no-operation instead of illegal.

FIGURE 3-5. SVC 2 Function Codes

3.5.1 Code 1 - Pause

This call is used to place the program in a suspended state. A PAUSE message is issued to the system console or log device. If the operator enters a CONTINUE command at the system console device, the program is restarted at the instruction immediately following the supervisor call. The parameter block format is as follows:

```
            ALIGN    ADC
PARBLK      DB       0,1   (no options)
```

3.5.2 Code 2 - Get Storage

This call gives a program a way to provide temporary storage locations for certain subroutines it may call, in particular FORTRAN Run-Time Library subroutines. This call does not increase the size of the program's memory allocation, but obtains locations from the program's current allocation. (See Section 1.4 on Memory Management). The parameter block format is as follows:

```
            ALIGN    ADC
PARBLK      DB       OPTION,2     DESIRED OPTION
            DC       H'REG'       ADDRESS REGISTER
            DC       F'SIZE'      NUMBER OF BYTES
```

Options allowed are:

X'00'	GET specified number of bytes
X'80'	GET all allocated storage

Note that the size parameter must be a fullword value; the register parameter must be a halfword value.

If option X'00' is specified, the system adjusts the system parameter, UTOP, upwards by the number of bytes requested. The starting address of the storage obtained is returned in the designated register. Subsequent calls with this option obtain new areas.

This call need not be used only on behalf of subroutines but the user program itself may find it a simple way to perform certain stack-manipulation operations. If option X'80' is specified, the system sets the system parameter UTOP equal to the system parameter CTOP+2 thus making available all of the task's current allocation. The starting address of the storage obtained is returned in the designated register and the number of bytes obtained is placed into the parameter block's SIZE parameter.

NOTE

If more storage is requested than is currently available, an address of zero is returned.

3.5.3 Code 3 - Release Storage

This call is the inverse of the GET STORAGE call (Code 2). It releases storage previously obtained. Storage is released on a last-in-first-out basis. The format of the parameter block is as follows:

```
          ALIGN    ADC
PARBLK    DB       0,3          NO OPTIONS
          DC       F'SIZE'      NUMBER OF BYTES TO BE RELEASED
```

Note that the size parameter must be a fullword value.

This call does not reduce the program's memory allocation. The pointer UTOP is adjusted downwards by this call, but not below UBOT. In Fullword mode, fullword alignment of the parameter block introduces a halfword of fill between the OPTION/CODE and SIZE fields.

3.5.4 Code 4 - Set Status

This call allows the user to modify the Arithmetic Fault (AF) Interrupt Enable bit and the Condition Code (CC) of the PSW. Two options are provided: the first option specifies that all allowable bits be modified; the second option specifies that only the Condition Code be modified. In Fullword mode, both the Halfword and Fullword Condition Codes are set. The format of the parameter block is:

```
          ALIGN    ADC
PARBLK    DB       OPTION,4       DESIRED OPTION
          DB       AF,CC          NEW STATUS, CONDITION CODE
```

The options allowed are:

X'00'	Modify Status and Condition Code
X'80'	Modify Condition Code only

The AF byte parameter indicates how arithmetic faults are to be handled and the valid combinations are:

X'00'	Disable Arithmetic Faults
X'10'	Enable Arithmetic Faults
X'14'	Enable Arithmetic Faults (halfword mode only)
X'04'	Enable Arithmetic Faults (halfword mode only)

The CC byte parameter has the form:

X'0x' where x is to replace the Condition Code.

3.5.5 Code 5 - Fetch Pointer

This call returns to the user the address of the table of parameters maintained by the system. The format of the parameter block is as follows:

```
            ALIGN    ADC
PARBLK      DB       0,5        NO OPTIONS
            DC       H'REG'     ADDRESS REGISTER
```

The address of the table of parameters is returned in register REG. The first three fullwords in this table are:

CTOP — Address of the last halfword in the task's allocated memory. The task must not attempt to access locations above this address. This value is modified by the EXPAND and CONTRACT ALLOCATION SVCs (SVC 2 Codes 20 and 21).

UTOP — Address of the first halfword following the user task program space. This value is the lowest available address for more storage and is modified by the GET STORAGE and RELEASE STORAGE SVCs (SVC 2 Codes 2 and 3).

UBOT — Address of the lowest location in the user task space. The task must not attempt to access locations below this address. System code occupies locations below UBOT.

The items in the system table following are unimportant to user tasks and are not defined here.

NOTE

The user should not attempt to modify the contents of CTOP, UTOP or UBOT by directly storing into these locations. CTOP and UTOP should be modified by using the appropriate SVC's; UBOT should not be modified at all. Failure to abide by this restriction may cause system failure.

3.5.6 Code 6 - Unpack Binary Number

This call performs the translation of a binary number contained in the user General Register Zero into ASCII hexadecimal or decimal format. The converted data is stored in a buffer pointed to in the parameter block. The format of the parameter block is as follows:

```
            ALIGN    ADC
PARBLK      DB       OPTION,6     DESIRED OPTION
            DC       A(DEST)      POINTER TO DESTINATION
```

where the A(DEST) parameter must be a fullword address.

Options recognized are as follows:

X'00' + N	Convert to Hexadecimal
X'80' + N	Convert to Decimal
X'C0' + N	Convert to decimal, suppress leading zeros
X'40' + N	Convert to hexadecimal, suppress leading zeros

Where 'N' is the length of the buffer supplied at DEST, in bytes. If 'N' = 0, a value of 4 is assumed.

The converted number is right-justified in the buffer so that the least significant digit occupies the last byte (highest address) in the buffer. If the number to be converted exceeds the buffer length, the most significant bytes are lost. If suppression of leading zeros is specified, the number is stored in the user buffer right-justified, and the remaining characters, if any, are filled with blanks.

The number in General Register Zero is considered to be an unsigned 32-bit constant.

N must be less than or equal to 63(X'3F'). It should be noted that 10 is sufficient for decimal, and 8 for hexadecimal.

Fullword alignment of the parameter block introduces a halfword of fill between the OPTION and A(DEST) fields.

3.5.7 Code 7 - Log Message

This call provides access from the user task to the system console (or log device). It gives the user a means of outputting a message with the assurance that it goes to the console, regardless of device assignments that may be in force. A number of options are provided, and the parameter block may take on two forms, Direct Text and Indirect Text. The format is as follows:

Direct Text

```
            ALIGN     ADC
PARBLK      DB        OPTION,7      DESIRED OPTION
            DC        H'LENGTH'     LENGTH OF MESSAGE IN BYTES
            DC        C'TEXT'       TEXT OF MESSAGE (DIRECT)
```

Indirect Text

```
        ALIGN     ADC
PARBLK  DB        OPTION,7      DESIRED OPTION
        DC        H'LENGTH'     LENGTH OF MESSAGE IN BYTES
        DC        A(TEXT)       ADDRESS OF MESSAGE TEXT (INDIRECT)
```

Options that are supported are as follows:

X'00' Direct Text, ASCII Formatted Message

X'80' Direct Text, ASCII Image Message

X'40' Indirect Text, ASCII Formatted Message

X'C0' Indirect Text, ASCII Image Message

The meaning of the Formatted/Image Message option depends on the driver, but its principal effect on a TTY or CRT is to suppress the Carriage Return/Line Feed sequence at end of the line if Image mode is selected. Note that the length of the text parameter is a halfword value. Indirect text may start on byte boundary, but its address is a fullword parameter.

A Log Message request is treated as I/O Proceed. If the console device is not busy, the task's message is transferred to a system buffer and the task proceeds concurrently with the message output. The task may then use the same parameter block with a different message and immediately issue another Log Message request. At this point, the task is suspended from execution until the previous log message is complete and the console becomes not busy.

3.5.8 Code 15 - Pack Numeric Data

This call is the inverse of the UNPACK BINARY NUMBER (SVC 2, Code 6) call and translates ASCII hexadecimal or decimal character strings to binary numbers. The format of the parameter block is as follows:

```
            ALIGN    ADC
PARBLK      DB       OPTIONS,15     DESIRED OPTION
            DC       H'REG'         REGISTER NUMBER HOLDING ADDRESS
```

Options recognized are:

X'00'	Hexadecimal
X'80'	Decimal
X'40'	Hexadecimal, skip leading blanks
X'C0'	Decimal, skip leading blanks

The result is returned in the user General Register 0. The register specified in the parameter block, halfword parameter REG, must point to the ASCII string that could not be converted (non-numeric if decimal is specified; other than 0-9 and A-F if hexadecimal is specified. The Fullword Condition Code is set to reflect the number of characters processed. If no characters could be processed, Register 0 is set to 0 and the CC 'L' bit is set (CC = 1). If more than eight hexadecimal characters were processed, only the least significant eight are reflected in Register 0 and the CC 'V' bit is set (CC = 4). If the decimal number processed was greater than $2^{32}-1$, the number returned in Register 0 is the actual number module 2^{31} and the CC 'V' bit is set (CC = 4). In all other circumstances, the CC is set to zero. Decimal arguments are treated as unsigned numbers. The returned Halfword CC is undefined.

NOTE

2^{32} = 4,294,967,296

Pack Numeric Data is a useful command processing function. See examples of its use in the OS/32 User Guides Manual, Publication Number 29-393.

3.5.9 Code 16 - Pack File Descriptor

This call is used to process a File Descriptor that is expressed in standard command syntax (VOLN:FILENAME.EXT) and place the result in a receiving area in standard SVC 7 parameter block format. This call is particularly useful if the name of the system volume is unknown.

The receiving area must be 16 bytes long and may be the name field of an SVC 7 parameter block. The format of the parameter block is:

```
            ALIGN    ADC
PARBLK      DB       OPTIONS,16      DESIRED OPTION
            DC       H'REG'          ASCII STRING ADDRESS REGISTER
            DC       A(DEST)         ADDRESS OF RECEIVING AREA
```

NOTE

The receiving area must be aligned on a fullword boundary.

Options recognized are:

X'00'	Default system volume
X'40'	Default system volume, skip leading blanks
X'80'	No default
X'C0'	No default, skip leading blanks

The register specified in the parameter block, halfword parameter REG, must point to the ASCII string to be processed. This pointer is returned pointing to the first character after the end of the File Descriptor. The Fullword Condition Code is set to reflect the file name processing. If a syntax error was detected, the 'V' flag is set (CC = 4). In this case, the contents of the receiving area is undefined. If no volume name (VOLN) was present in the string, the CC 'L' flag is set (CC = 1); the contents of the receiving area in this case depend on the selected option. If default system volume is selected, the current system volume name is placed in the first four-byte field of the receiving area. If 'no default' is selected, the first four bytes of the receiving area remain unchanged. If no extension is present in the string, the CC 'C' flag is set (CC = 8) and the last four bytes of the receiving area are set to ASCII blanks. If the ASCII string being processed contains less than the maximum numbers of characters for any field, the excess characters are blank-filled. File Descriptor scan is stopped by any non-alphanumeric except for colon (:) and period (.). Except as previously explained, the Fullword CC is otherwise zeroed. The returned Halfword CC is undefined.

NOTE

See note regarding command processing under Pack Numeric Data.

3.5.10 Code 17 - Scan Mnemonic Table

This call presents a pointer to an input string of ASCII characters and the address of a standard format mnemonic table.

A mnemonic table is composed of strings of 7-bit ASCII bytes, separated by a null byte (X'00'). The end of the table is signified by two consecutive null bytes. Mnemonics may contain only alphabetic characters, except for the first character, which may be any character except a null byte. All required bytes of a mnemonic word in the table must be flagged with B 0 = 1; non-required bytes are flagged with Bit 0 = 0.

This call scans the mnemonic table for a match on the input string. If a match occurs, the string pointer is returned pointing to the byte in the input string following the match, the index value of the matched mnemonic (i.e., its position in the mnemonic table) is returned, and the Fullword Condition Code is zeroed. If no match is found, the string pointer is returned unchanged, the index value returned is -1, and the 'V' bit of the Fullword CC is set (CC = 4). The returned Halfword CC is undefined.

The parameter block format is:

```
          ALIGN    ADC
PARBLK    DB       0,17                    NO OPTIONS
          DB       REG1, REG2              INPUT AND INDEX REGS.
          DC       A(MNEMONIC TABLE)
```

Where REG1 byte specifies the register which contains the address of the input string and REG2 byte specifies the register in which the table index is to be placed. A(MNEMONIC TABLE) is the fullword address of the mnemonic table to be scanned.

NOTE

The returned index value counts from zero, i.e., it is zero if the match is found on the first table entry, one if the second entry, two if the third entry, etc.

See note regarding command processing under Pack Numeric Data.

3.5.11 Code 18 - Move ASCII Characters

This routine is used to move ASCII characters from an input string to a target string. The number of characters moved is controlled by length or by the occurrence of an ending character in the ASCII string. The parameter block format is:

```
          ALIGN    ADC
PARBLK    DB       OPTION,18     OPTIONS
          DB       REG1,REG2     INPUT AND OUTPUT POINTERS
          DC       A(ECSTRING)   ADDRESS OF ENDING CHAR. STRING
```

Where REG1 byte specifies the register pointing to the input ASCII string, and REG2 contains a pointer to the receiving area. The ending character string is a string of ASCII characters, any one of which halts the move operation. The first byte of the ending character string must contain a binary count of the

number of ending characters.

Options recognized are:

X'00' + N No Ending Characters

X'80' + N Use Ending Characters

Where N is the maximum number of characters to be moved, limited to X'7F' (decimal 127).

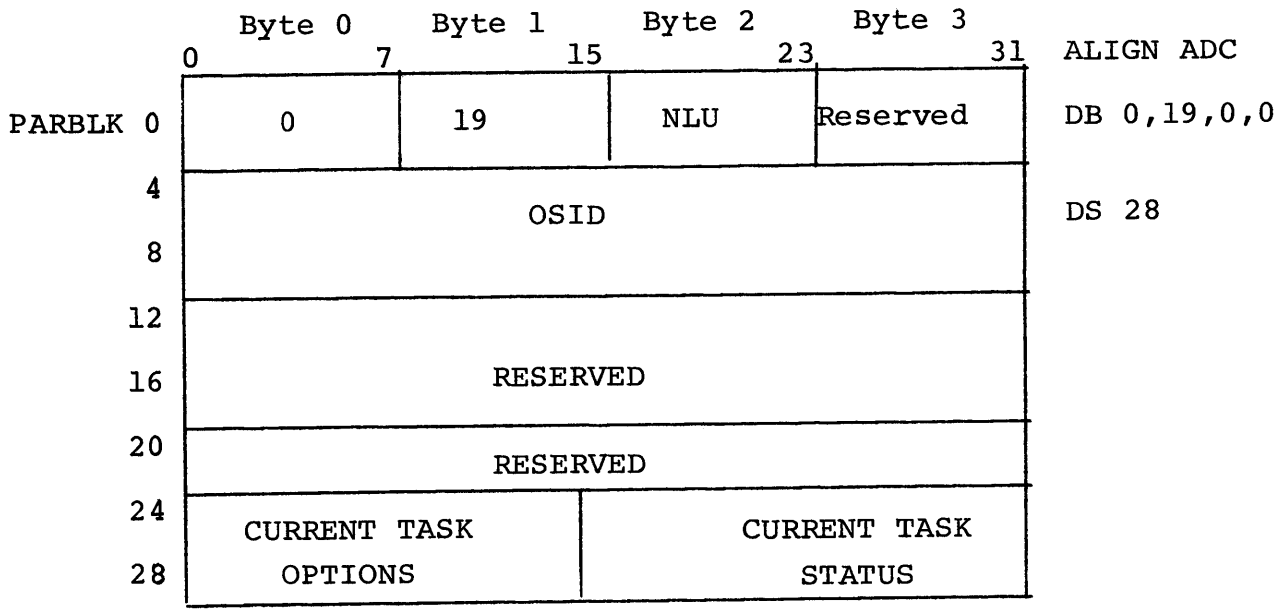
If the ending character option is selected and no ending character is found, the 'V' flag is set (CC = 4) in the Fullword Condition Code, otherwise it is zeroed. The input and output string pointers are returned pointing to the byte following the last byte moved in the input and output areas respectively. The returned Halfword CC is undefined.

NOTE

See note regarding command processing under Pack Numeric Data.

3.5.12 Code 19 - Peek

This call allows a task to extract certain system and task-dependent information from OS/32. The information is moved to the user's parameter block, defined in Figure 3-6.



Note: The first 2 bytes are the OPTIONS (no options) and CODE fields. The other fields are supplied by the system.

Figure 3-6. PEEK Parameters

NLU is the number of Logical Units available to the task.

OSID is the form OS32STRR, an eight-byte ASCII field suitable for printing. RR is the revision level of the Operating System.

The TASK OPTIONS and TASK STATUS fields, used by system programs, reflect the current state of the user program (see Section 3.2).

NOTE

The PEEK Parameter block must be 32 bytes long or information following it is overwritten.

3.5.13 Code 20 - Expand Allocation

This call causes memory to be added to the task's allocation in blocks of 256 bytes. Memory is always allocated contiguously upwards from the top of the task's previous allocation (as long as there is sufficient remaining physical memory for the requested expansion). The format of the parameter block is as follows:

```
          ALIGN    ADC
PARBLK    DB       OPTION,20     DESIRED OPTION
          DC       H'N'          NUMBER OF 256-BYTE BLOCKS
                                    REQUIRED
```

Options are:

X'00' Expand number of blocks specified by halfword parameter N.

X'80' Obtain all of available memory and return number of blocks in N.

This call returns a Fullword Condition Code of zero if the call was successful; the returned Fullword CC is non-zero if the call was unsuccessful. The returned Halfword CC is undefined.

This call modifies CTOP, permitting further use of SVC 2 Code 2 (GET STORAGE). The call is rejected with the 'G' bit of the Fullword CC (CC = 2) if not enough physical memory is available at the time of a call with options X'00'. With options X'80', N is returned equal to zero if no additional memory is available.

NOTE

Too great an allocation for the program may prevent the assigning of direct access files because the Operating System uses the area between the top of the task's allocation (CTOP) and the top of memory (MTOP) for File Control Blocks (FCBs). See Chapter 4 for an example of using the Expand call in conjunction with Get Storage to obtain memory in an orderly manner.

3.5.14 Code 21 - Contract Allocation

This call is the inverse of the EXPAND call. It causes memory in the task's allocation to be released, from the top down, in 256 byte blocks. The format of the parameter block is:

```
            ALIGN    ADC
PARBLK      DB       0,21     (no options)
            DC       H'N'     NUMBER OF 256-BYTE SECTORS TO BE
                                RELEASED
```

This call modifies CTOP. This call is rejected if it would cause CTOP to be less than UTOP. Therefore, sufficient SVC 2 Code 3 (RELEASE STORAGE) calls should be made prior to this call. If the call was successful, the Fullword Condition Code (CC) returned is zero. If the call is rejected, the Fullword CC 'L' bit (CC = 1) is set. The returned Halfword CC is undefined.

3.6 SVC 3 - END OF TASK (EOT)

This call allows a task to terminate itself in an orderly fashion. Its format is:

```
SVC 3, N    EOT
```

There is no parameter block associated with this call. Instead, the resultant parameter block address of the SVC Instruction is treated as a binary constant and it, truncated to 16 bits, replaces the System Return Code used by the Command Substitution System (see Chapter 2).

If the task issuing this call has I/O in progress at the time the call is made, the I/O is terminated. All assigned Chained files on direct-access devices are checkpointed (see Section 3.3.5). All LUs remain assigned exactly as before the SVC 3 was executed.

3.7 SVC 5 - OVERLAY CALL

This call permits a task to fetch an overlay from a specified Logical Unit (LU). See Chapter 4 for examples of use of SVC 5.

The format of the call is as follows:

```
            SVC     5,PARBLK
             :
             :
            ALIGN   ADC
PARBLK      DC      C'OVERLAY'    EIGHT-CHARACTER OVERLAY ID
            DB      X'00'         STATUS BYTE
            DB      X'xx'         OPTIONS BYTE
            DC      H'LU'         LOGICAL UNIT (HALFWORD)
```

Options recognized are:

X'01' Load from LU without positioning

X'04' Load from LU after Rewind

Status returned is:

X'00' Overlay loaded successfully

X'10' Load failed: for various possible reasons

X'40' Load failed: Overlay would not fit in Allocated memory

The 8-character overlay ID field of SVC 5 is ignored in OS/32-ST, but the field is required for compatibility with other INTERDATA Operating Systems.

Overlays must be linked with the root program by establishing them with the OS/32 Library Loader (see examples in Chapter 4).

The calling program is suspended until the overlay is loaded. If the overlay is successfully loaded, the root program may Branch and Link to it as a subroutine.

Overlays should not call other overlays, otherwise the results may be unpredictable.

The task should use caution with the SVC 2, Code 2 (GET STORAGE) call when using overlays. Figure 3-7 shows a typical memory map after loading a main program which uses overlays. The area shown between UTOP and OBOT is allocated when the program was established with the OS/32 Library Loader. Get Storage requests are satisfied from this area. Overlays always load at OBOT and if a Get Storage call causes UTOP to exceed OBOT, some data may be overlaid. The size of this Get Storage area may be zero if the task does not issue an SVC 2, Code 2.

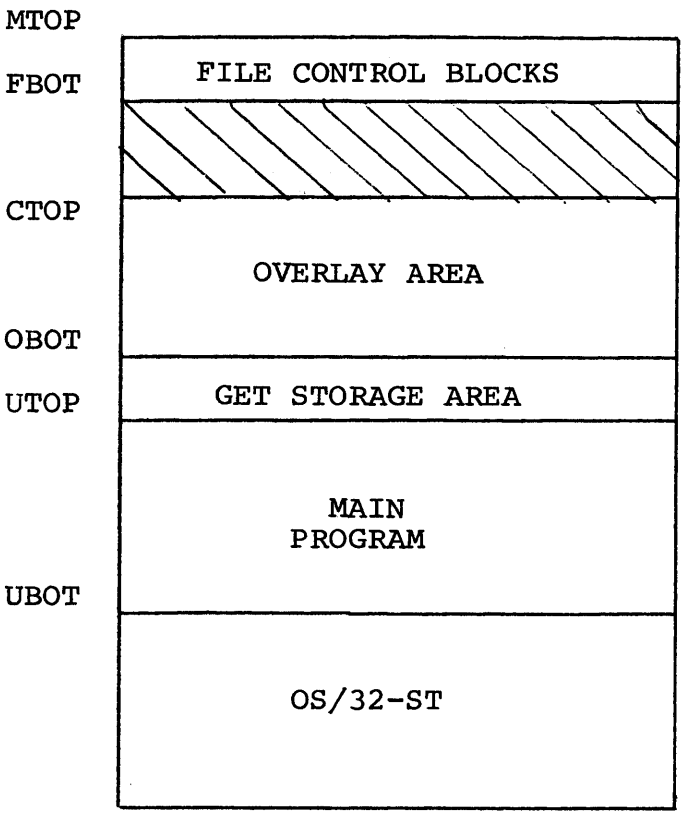


FIGURE 3-7.

Memory Map Showing Overlay Area

3.8 SVC 7 - FILE HANDLING SERVICES

This call gives the user task a facility whereby files on direct-access devices may be created, assigned to Logical Units (LUs), closed, renamed, reprotected, checkpointed and deleted. In addition, non-direct access devices may be assigned to LUs and deassigned through this call. I/O is not performed through this call; the user must use SVC 1 for that purpose. The format of the SVC 7 parameter block, which must be on a Fullword boundary, is illustrated in Figure 3-8.

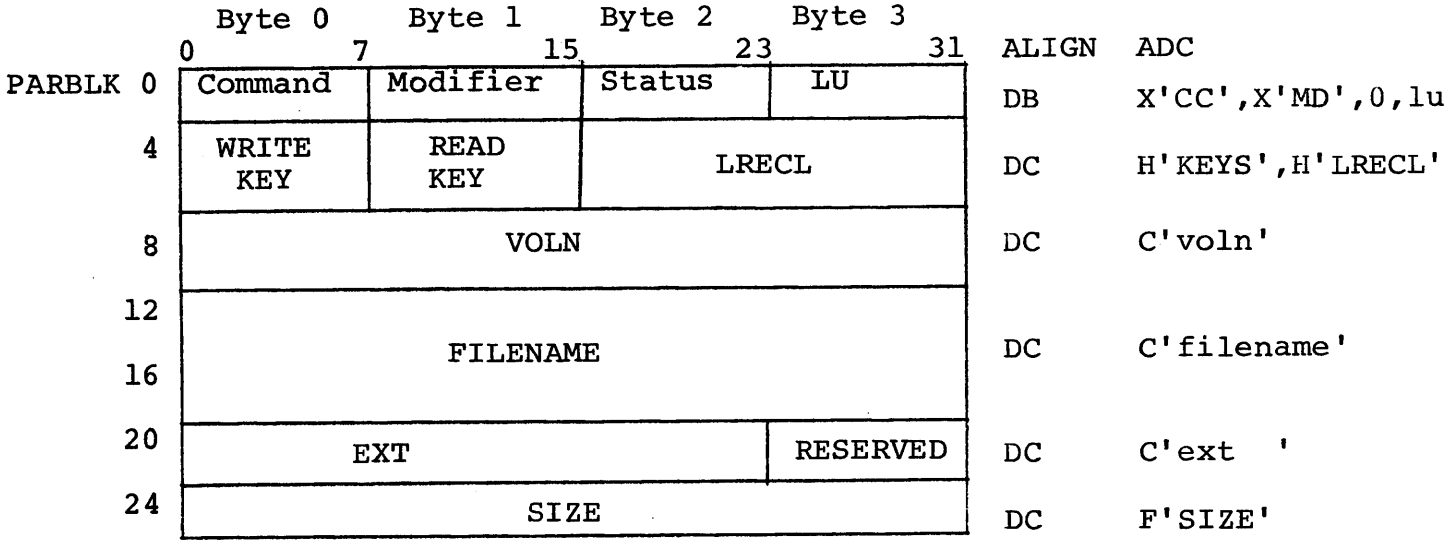


Figure 3-8. SVC 7 Parameter Block

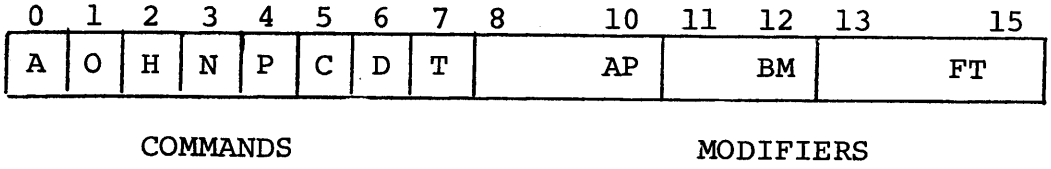
Several examples of an SVC 7 parameter block are given in Chapter 4.

Not all the elements of this parameter block are required for all variations of the SVC 7 call. The requirements are discussed in detail under the specific options following.

3.8.1 SVC 7 Parameter Block Fields

COMMAND/MODIFIER

The format of the Command/Modifier Halfword is shown in Figure 3-9.



The functions to be performed are specified in the command byte (Bits 0-7). They are as follows:

- A (Bit Ø) Allocate; requires File Type (FT) field as modifier
- O (Bit 1) Assign; requires Access Privilege (AP) and Buffer Management (BM) fields as modifiers.
- H (Bit 2) Change access privileges; requires AP field as modifier.
- N (Bit 3) Rename.
- P (Bit 4) Reprotect.
- C (Bit 5) Close.
- D (Bit 6) Delete.
- T (Bit 7) Checkpoint.

all bits zero = Fetch Attributes

FIGURE 3-9. SVC Command/Modifier Halfword

The SVC 7 functions are sequentially processed from left to right, if more than one command bit is set.

Bits 8 - 15 are modifiers. They are as follows:

Bits 8 - 10 specify Access Privileges, encoded as follows:

```
000 = SRO      (Shared Read Only)
001 = WRO      (Exclusive Read Only)
010 = SWO      (Shared Write Only)
011 = EWO      (Exclusive Write Only)
100 = SRW      (Shared Read-Write)
101 = SREW     (Shared Read, Exclusive Write)
110 = ERSW     (Exclusive Read, Shared Write)
111 = ERW      (Exclusive Read-Write)
```

Bits 11 - 12 specify Buffer Management, encoded as follows:

```
00 =  Default buffer management method
01 =  Unbuffered Physical
10 =  Buffered Logical
11 =  Reserved, considered illegal
```

The Default Buffer Management method is unbuffered physical for Contiguous files and buffered logical for Chained files.

Bits 13 - 15 specify File Type, encoded as follows:

```
000 = Contiguous
001 = Chained
010 =
...   } Reserved, considered illegal
111 =
```

On a Fetch Attributes call, the modifier field is not used in the call, and instead the Device Code is returned in this field.

Error Status

The interpretation of the status byte depends upon the commands specified in the call and is defined under the description of each command. A status of zero always means the desired options were performed without error. A summary of all possible error codes is given in Figure 3-10 for reference (hexadecimal).

Hexadecimal	Meaning
00	No error, the requested functions are complete
01	Illegal function, illegal file type or buffer management
02	LU Error; illegal LU
03	Volume Error; no such volume in system
04	Name Error; mismatch on FILENAME.EXT field
05	Size Error; erroneous LRECL or SIZE field
06	Protect Error; erroneous protection keys
07	Privilege Error; unable to obtain requested privilege
08	Buffer Error; no room in system for File Control Blocks (FCBs) or buffers
09	Assignment Error; LU not assigned
0A	Type Error; non-direct access device, or device offline
0B	File Descriptor Error; illegal syntax
80-FF	I/O Error; interpreted as SVC 1 Status Byte

FIGURE 3-10. Interpretation of SVC 7 Status Byte

These errors are listed in the order in which the system checks for them. The first error detected causes the SVC to return. If multiple functions were specified (e.g., Allocate and Assign) some functions may have been properly performed (always in left-to-right sequence).

LU

This byte defines the Logical Unit used for all the SVC 7 functions except Allocate and Delete.

WRITE KEY AND READ KEY

Protection keys for direct-access files and devices are specified in this halfword. These keys are required for the Allocate, Assign, Reprotect and Delete functions. This field is used by the Fetch Attributes call to return the device or file attributes.

LRECL

This halfword field, on an Allocate call must contain the logical record length for a buffered file. On a Fetch Attributes call, the logical record length of a file or physical record length of a device is returned in this field. LRECL is not used for the other functions.

VOLUME ID or DEVICE MNEMONIC (VOLN)

This four-byte field identifies the volume (e.g., removable disc cartridge) if it is a direct access device, or the device mnemonic if it is a non-direct access device. This field in ASCII characters is required for the Allocate, Assign, Delete and Fetch Attributes functions.

VOLN together with FILENAME and EXT fields identify a File Descriptor. VOLN is returned by the system on a Fetch Attributes call.

FILENAME

This eight-byte field identifies the file on a direct access device; it is not required for a non-direct access device. The user must specify filename in ASCII characters for Allocate, Assign, Rename and Delete calls. FILENAME is returned by the system on a Fetch Attributes call; it is blank for a device.

EXTENSION

This three-byte ASCII field identifies the file type (e.g., OBJ, TSK, CSS, etc.), on a direct access device. It is treated as an extension of, and required under the same conditions as the FILENAME. The byte following this field is reserved for future expansion and is ignored.

SIZE

This fullword field on the Allocate call, must contain the file size in sectors for a Contiguous file or the physical block size in sectors for a Chained file. On a Fetch Attributes call, this field is used to return the current size of a direct access file; SIZE is not used for a non-direct access device. Note that the full parameter block should be specified for the Fetch Attributes call or the system may overwrite other information.

3.8.2 SVC 7 Functions

ALLOCATE

The Allocate function reserves space on a direct-access device and in the directory for the specified file type (FT). In the

case of Chained files, only a directory entry is made. The protect keys are entered in the directory. The required parameters are FT, KEYS, LRECL, VOLN, FILENAME, EXT and SIZE.

Applicable error codes are:

01	Illegal Function; an illegal file type was specified
03	Volume Error; the specified volume was either not mounted or is not a direct-access device
04	Name Error; the specified file name already exists on the specified volume
05	Size Error; if a Contiguous file: there is not sufficient contiguous space on the specified volume to allocate a file of the specified size, if a Chained file, the specified physical block size exceeds the limit established at SYSGEN time
0A	Type Error
0B	File Descriptor Error

ASSIGN

The ASSIGN function establishes a logical connection between a file (or device) and the task through a specified Logical Unit, under a given access privilege (AP) and using a given Buffer Management (BM) technique. The call proceeds as follows: first the access privilege is examined to determine which protect key to check. The proper key is then checked against the keys in the file directory. The BM field is checked to see if the required Buffer Management technique is valid for the type of file being assigned. The file is then assigned according to the requested access privileges. If SWO or EWO (write only) is specified, the file is positioned at its logical end (records are appended). Otherwise, the file is positioned at the beginning (record number = zero). Only the keys are checked for given access privilege, for non-direct access devices.

The required parameters are AP, BM, LU, KEYS, VOLN, NAME and EXT; BM, NAME and EXT fields are not used for non-direct access devices.

Applicable error codes are, in order of decreasing procedence:

01	Illegal function; invalid BM field
02	LU Error; illegal LU or LU already assigned
03	Volume Error; no such volume
04	Name Error; no such name on given volume
06	Protect Error; mismatch on protection keys
07	Privilege Error; requested privilege may not be granted
08	Buffer Error; no room for FCB or buffers
80-FF	I/O Error

NOTE

A buffered file requires two system buffers to be allocated in memory, each equal to the physical block size of the file.

For Chained files, only BM = 00 or 10 is currently valid;

For Contiguous files, only BM = 00 or 01 is currently valid.

CHANGE ACCESS PRIVILEGES

This function allows the user to change the current access privileges of a file or device which is assigned. Only two parameter fields in the SVC 7 Parameter Block are required, the LU and AP portion of the modifier field.

The requested new access privilege cannot, however, change the I/O mode which was established when the file was originally assigned. For example, if the file was assigned for ERO, no

access privilege requiring write access is allowed. Figure 3-11 shows all valid access privilege changes.

Entries of Y denote a valid request; N denotes an invalid request.

If an error is encountered while processing this request, the file remains assigned with its original access privilege.

Applicable error codes are:

02 LU Error; illegal LU
07 Privilege Error; new Privileges cannot be granted
09 Assignment Error; LU not assigned

CHANGE TO / CHANGE FROM	SRO	ERO	SWO	EWO	SRW	SREW	ERSW	ERW
SRO	Y	Y	N	N	N	N	N	N
ERO	Y	Y	N	N	N	N	N	N
SWO	N	N	Y	Y	N	N	N	N
EWO	N	N	Y	Y	N	N	N	N
SRW	Y	Y	Y	Y	Y	Y	Y	Y
SREW	Y	Y	Y	Y	Y	Y	Y	Y
ERSW	Y	Y	Y	Y	Y	Y	Y	Y
ERW	Y	Y	Y	Y	Y	Y	Y	Y

FIGURE 3-11. Valid Access Privilege Changes

RENAME

This function permits the name of an assigned file to be changed. The file must currently be assigned for ERW. The required parameters are LU, FILENAME and EXT. The given LU must be assigned to a direct-access file unless the caller is a system Executive task which may rename non-direct access devices. The volume name field of the parameter block is ignored. The specified FILENAME.EXT replaces the previous FILENAME.EXT in the directory if the Rename function is successful.

Applicable error codes are:

02	LU Error; illegal LU
04	Name Error; new name already exists on given volume
07	Privilege Error; file not open for ERW (Exclusive Read/Write)
09	Assignment Error; LU not assigned
0A	Type Error; LU for non-direct access device
0B	File Descriptor Error
80-FF	I/O Error

REPROTECT

This function permits the protection keys of an assigned file to be changed. The required parameters are Keys and LU. The given LU must be assigned to a direct-access file, which must currently be assigned for ERW (unless the caller is a System Executive task which may modify the protection keys of non-direct-access devices). If either the specified read key or write key is equal to X'FF', that key is ignored (unless the caller is an Executive task). If the call is rejected, the previous keys remain unchanged.

Applicable error codes are:

02	LU Error; illegal LU
07	Privilege Error; not assigned for ERW
09	Assignment Error; LU not assigned
0A	Type Error; LU for non-direct access device
80-FF	I/O Error

CLOSE

This function discontinues an assigned logical connection between a task and a file or device. LU is the only required parameter. The specified LU is deassigned. Files assigned for write access have any partially filled buffers written to the file by the CLOSE call.

Applicable error codes are:

02	LU Error
09	Assignment Error
80-FF	I/O Error

DELETE

For a Delete function, the VOLN, FILENAME, EXT, and Keys parameters must specify a direct-access file which is not currently assigned. If these conditions are met and both read and write keys match, the file is deleted from the directory of its volume.

Applicable error codes are:

03	Volume Error; no such volume
04	Name Error; file name does not exist on volume
06	Protect Error; invalid protection keys
07	Privilege Error; file not closed
0B	File Descriptor Error
80-FF	I/O Error

CHECKPOINT

The Checkpoint function permits the user to flush system Buffer Management buffers and to update the directory entry for a file on a given LU. LU is the only required parameter. This function, in OS/32-ST, is only recognized for Chained files; requesting a checkpoint for a Contiguous file or for a non-direct access device has no effect.

The user may wish to employ Checkpointing after sensitive data is added to a buffered file because the logical blocking of data is memory in system buffers leaves the file vulnerable. The integrity of the data can be preserved on disc by Checkpointing. In case of system failure, all data on Chained files up to the latest Close or Checkpoint operation is recoverable; data appended after the most recent Checkpoint is lost. Checkpoint differs from a Close/Assign sequence in that no repositioning is performed and that file name, access privileges, and keys need not be specified.

The applicable error codes are:

02	LU Error
09	Assignment Error
80-FF	I/O Error

FETCH ATTRIBUTES

Certain programs may require, for proper operation, the knowledge of the physical attributes of the device or file associated with a given LU. For example, it might be desirable to know if random access is possible or if the device is rewindable. The Fetch Attributes function gives the user access to this information.

Only one status error response is possible:

02 LU Error; illegal LU

Various fields within the SVC 7 parameter block are redefined for purposes of this call. When the command field is set to zero, indicating a Fetch Attributes call, the only required parameter is the Logical Unit. The system returns information in fields Keys, LRECL, File Descripter, SIZE and Modifier byte.

The Write Key/Read Key halfword is redefined to receive an attributes halfword as given in Figure 3-12. Any bit set means the device or file supports the corresponding attribute.

BIT	ATTRIBUTE
0	RESERVED
1	SUPPORTS READ
2	SUPPORTS WRITE
3	SUPPORTS BINARY
4	SUPPORTS WAIT I/O
5	SUPPORTS RANDOM
6	SUPPORTS UNCONDITIONAL PROCEED
7	SUPPORTS IMAGE
8	RESERVED
9	SUPPORTS REWIND
10	SUPPORTS BACKSPACE RECORD
11	SUPPORTS FORWARD SPACE RECORD
12	SUPPORTS WRITE FILEMARK
13	SUPPORTS FORWARD SPACE FILEMARK
14	SUPPORTS BACKSPACE FILEMARK
15	RESERVED

FIGURE 3-12. SVC 7 Device Attributes Halfword

The LRECL field is set by the system to the physical record length associated with the device (e.g., 80 for a card reader, 120 or 132 for a line printer) if the record length is fixed; these two bytes are set to zero for a variable record length device (e.g., mag tape).

A TTY or CRT, which is in the strict sense a variable-record length device, normally has LRECL set as though it were a fixed-record length device. This is because such a device is normally used in a fixed-record length method. Furthermore, a Contiguous direct-access file is considered to have a fixed record length, which is the logical record length chosen for that file at the time of its allocation.

The volume name, file name, and extension (VOLN:FILENAME.EXT) for a named file, or the device mnemonic for a non-direct access device is returned in the File Descriptor portion of the parameter block. The program may use this information, e.g., to provide the operator with intelligent information in case of error, to generate the name of a new output file by transformation on the name of the input file, etc.

The current size of a direct-access file is returned in the SIZE field; SIZE is unchanged for non-direct access devices. Number of logical records is returned for a Chained file; number of sectors is returned for a Contiguous file.

The MODIFIERS byte is set to indicate the file or device type. A sample of device codes is given in Figure 3-13. The codes for

all supported devices is given in the OS/32 Series General Purpose Driver Manual, Publication Number 29-384.

CODE (DECIMAL)	FILE OR DEVICE TYPE
0	Contiguous File
1	Chained File
16	Model 33 TTY, Keyboard/Printer
17	Model 35 TTY, Keyboard/Printer
18	Nonediting CRT
64	800 BPI Magnetic Tape
65	1600 BPI Magnetic Tape
66	INTERTAPE Cassette
80	High Speed Paper Tape Reader/Punch
81	Model 33 TTY, Reader/Punch
82	Model 35 TTY, Reader/Punch
96	CPM Card Reader
112	Centronics line printer
114	High Speed line printer
255	NULL DEVICE

FIGURE 3-13. Example Device Codes

CHAPTER 4

EXAMPLES

4.1 INTRODUCTION

This chapter gives examples of operator and program functions under OS/32-ST and familiarizes the user with program control techniques.

An in-depth treatment of some of the basic facilities in OS/32-ST can be found in the OS/32 User Guides Manual, Publication Number 29-393.

Sizing and performance information can be found in the OS/32-ST Program Configuration Manual, Publication Number 29-379.

4.2 OPERATION EXAMPLES

On all following examples of command and data entry, the characters '*' and '>' at the beginning of a line are output by the system as prompts to the operator.

4.2.1 Establishing Programs on Disc

The following procedure demonstrates how the user may establish a program on a disc file for convenient access. Assume the program exists as an object module on paper tape. (Standard record length for OS/32 object modules is 126 bytes).

First, a disc file (the system volume is assumed) must be Allocated to contain the program.

Enter:

```
*ALLOCATE PROGRAM.OBJ,CHAIN,126    Create file
*ASSIGN1,PTRP:                     Assign paper tape reader to LU1
*ASSIGN2,PROGRAM.OBJ               Open file to LU2
```

Next, the OS LIBRARY LOADER must be loaded, as it is used to copy the program to the file. Insert the OS/32 Library Loader tape in the paper tape reader and enter:

```
*LOAD PTRP:
*START
```

Now insert the program to be copied to disc in the paper tape reader and enter the following commands to the OS Library Loader.

```
>COPY 1,2
>END
```

Unless the program file is to be used at this time, it may be closed.

```
*CLOSE 2
```

Now, anytime the program in file PROGRAM.OBJ, is required in memory, it may be loaded via the following command:

```
*LOAD PROGRAM.OBJ
```

NOTE

Default protection keys (X'0000') and default access privileges (SRW, converted to SREW by the system for Chained Files) are used.

4.2.2 Assembling With Disc

This procedure demonstrates how to start the Assembler and assemble a user program. The system disc volume is assumed. Also assumed: the assembler is in a disc file named CAL.OBJ, the source is read from device CARD:, the object is written to file USER.OBJ, and a disc file is used for the scratch files. Assume only file CAL.OBJ exists on disc. A symbol table is not required; the CAL source contains NO COPY statements, and SQUEEZE is not requested. The following command sequence is required.

*ALLOCATE	SCRATCH1	Create scratch file for cross ref.
*ASSIGN	5,SCRATCH1	Assign scratch to LU5
*ASSIGN	6,NULL	Assign LU6 to Null Device
*ASSIGN	7,NULL	Assign LU7 to Null Device
*ASSIGN	8,NULL	Assign LU8 to Null Device
*ALLOCATE	SCRATCH	Create scratch file
*ASSIGN	4,SCRATCH	Assign scratch to LU4
*ALLOCATE	USER.OBJ,CHAIN,126	Create user program file
*ASSIGN	2,USER.OBJ	Assign binary object to LU2
*ASSIGN	3,PRINT:	Assign list device to line printer on LU3
*ASSIGN	1,CARD:	Assign source input to card reader on LU1
*LOAD	CAL.OBJ	Load the Assembler
*START		Start the Assembler

Both the scratch file and the binary object file are allocated as Chained files since the size of neither file is known in advance.

When the assembly is complete, the user may wish to:

*CLOSE 2	Leave object program on disc
*CLOSE 4	Close scratch file
*CLOSE 5	
*DELETE SCRATCH	Remove the scratch files on disc
*DELETE SCRATCH1	

Deleting the scratch files is not necessary. They can be re-assigned again after closing if desired to reuse the files; but they should first be CLOSEd or REWINDed.

4.2.3 FORTRAN Compiling with Disc

Assume the FORTRAN Compiler, OS Library Loader, OS Assembler, and the FORTRAN Run Time Library have been established on disc files using the procedure specified in Section 4.2.1. The user wishes to compile and execute a FORTRAN source program on cards. The FORTRAN compiler itself reads its input from LU1, outputs assembly language source on LU2, and prints output on LU3. The required sequence status as follows.

```
*ASSIGN 1,CARD:       Assign card reader as SOURCE input to LU1
*ASSIGN 3,PRNT:       Assign list device to line printer in LU3
*ALLOCATE USER.CAL,   Create file for assembly
*LOAD FORTRAN         Load FORTRAN compiler
*ASSIGN 2,USER.CAL    Assign output to LU2
*START                Start compiler
```

After the FORTRAN compiler completes, the Assembler must be run to assemble the data written to file USER.CAL by FORTRAN. This procedure is outlined in Section 4.2.2. Note, however, that LU1 has to be closed and reassigned to USER.CAL, and that LU2 has to be closed and reassigned to USER.OBJ (or to some other acceptable name).

Since most FORTRAN programs require routines from the Run Time Library, the OS Library Loader must be used to create the final executable task. Assuming the FORTRAN RTL is on a file named RTL, and the Library Loader is on a file named LIBLDR, start the Library Loader as follows:

```
*ASSIGN 6,RTL        Assign Run Time Library to LU6
*LOAD LIBLDR         Load Library Loader
*START               Start Loader
```

Now input the following commands to the loader:

```
BI   XXXX    Set load bias to some location above loader
RE   2       Rewind binary object file
LO   2       Load binary object file
ED   6       Edit with Run Time Library
END          End Library Loader
```

Any file assignments required for use by the FORTRAN program may now be entered. Following these assignments, the program may be started at the BIAS specified in the first command to the loader. Enter:

```
ST XXXX    Start Program
```

4.2.4 Building 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 SVC 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, the saving is achieved by sacrificing time. Even when disc is used as the overlay medium, it takes time to load an overlay. For this reason, overlays should be used for processing that is not time-critical.

The OS/32 Library Loader is used to establish overlays (see the 32-Bit Series Loader Description Manual, Publication Number 29-376). The command, OV, 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.

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 LU2 and the second via LU3 using the IFETCH routine in the Run Time Library. We begin by allocating disc files for the main program and the overlay subroutines, and assigning them to LUs, 1, 2, and 3, respectively. Note that standard object record size is 126 bytes. Assume the Library and Loader are on disc files RTL and LIBLDR, respectively:

```
*ALLOCATE  MAINP,CHAIN,126
*ALLOCATE  MAINP.OV1,CHAIN,126
*ALLOCATE  MAINP.OV2,CHAIN,126
*ASSIGN    1,MAINP
*ASSIGN    2,MAINPOV1
*ASSIGN    3,MAINPOV2
*ASSIGN    6,RTL              Assign Run Time Library to LU6
*ASSIGN    4,PTRP:            Assign paper tape reader to LU7
```

Now load and start the OS/32 Library Loader

```
*LOAD LIBLDR
*START
```

The following commands are read by the Library Loader.

```
>OUT  1     Set the out option for LU1
>LO   4     Load the main program from paper tape
>ED   6     Edit with the Run Time Library
>XOUT
>OV         Indicate an overlay to be linked
>OUT  2     Set the out option for LU2
>LI   4     Link first overlay from paper tape
>ED   6     Edit with the Run Time Library
>XOUT
>OV         Indicate another overlay is to be linked
>OUT  3     Set the out option for LU3
>LI   4     Link second overlay from paper tape
>ED   6     Edit with the Run Time Library
>XOUT       Terminate out option
>END        Terminate Library Loader
```

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

```
*CLOSE 1
*CLOSE 2
*CLOSE 3
*LOAD MAINP
```

Now make any LU assignments required for program execution and start the program

```
*START
```

4.2.5 Example CSS Files

1. This example shows a CSS file to load and run a program, assuming that the LU assignments are already made. If the program goes to Abnormal End, then a skip to $TERMJOB is initiated, otherwise, the file exits.

```
*RUN <PROGNAME>[,[<START PARA>],[<START ADDRESS>]]
*
$IFNULL @1; $SKIP; $ENDC;* IF NO PROGRAM SPECIFIED SKIP TO
*                          TERMJOB
LOAD @1.OBJ              ;* PROGRAM IS ASSUMED TO BE ON DISC
*                          FILE
*                          PROGNAME .OBJ
START @3,@2              ;* START PROGRAM, PASSING PARAMETERS
*                          IF SPECIFIED
$IFNE 0; $SKIP; $ENDC    ;* SKIP TO $TERMJOB IF PROGRAM
*                          GIVES ABNORMAL END
$EXIT
```

A file like this can be thought of as an extra command. If a library of such extra commands is created, it is good practice to include, as a comment at the start of each file, a statement of the command SYNTAX.

2. Example of a file used to create a direct access file. If the file already exists, no action is taken.

```
*CREATE  FILE-ID
*
$IFX @1; $EXIT; $ENDC;*     IF FILE-ID ALREADY EXISTS
*                           TAKE NO ACTION
ALLOCATE @1;*               ALLOCATE CHAINED FILE
*                           WITH DEFAULT CHARACTERISTICS
$EXIT
```

3. This example is included to demonstrate the versatility of CSS. A number of CSS-files are used, and they call each other to a depth of 3.

CSS-file MESSAGE

```
$COPY
*RETURN CODE = @@3, IF@@2 @@1 @1
$NOCOPY
$EXIT
```

This file is called by files True and False to list messages of the form

RETURN CODE = 0, IFE 0 OK

RETURN CODE = 255, IFL 0 OK

CSS-file TRUE

```
$JOB
SET CODE @3
$IF@2 @1                ;  *SEE NOTE
MESSAGE OK
$SKIP
$ENDC
MESSAGE FAILED
$TERMJOB
$EXIT
```

Note: This artificially constructed conditional should give true. If it does, a message is listed on the console confirming that the conditional is OK. If it doesn't, the Failed message is printed.

CSS-file FALSE

```
$JOB
SET CODE @3
$IF@2 @1                ;  *SEE NOTE
MESSAGE FAILED
$SKIP
$ENDC
MESSAGE OK
$TERMJOB
$EXIT
```

Note: This artificially constructed conditional should give false. If it does, a message is listed on the console confirming that the conditional is OK. If it doesn't, the Failed message is printed.

CSS-file CHECK

```
@4 @3, @2, @1;*      SEE NOTE
$EXIT
```

Note: This file is a linking file, it provides an elegant method of testing the return code conditional operations, $IFE, $IFNE, $IFL, $IFNL, $IFG and $IFNG.

A call of CHECK takes the form:

CHECK RETURN CODE , CONDITION , VALUE , ANSWER

where RETURN CODE is the value to which the return code is to be set,

where CONDITION determines the conditional to be tested, according to the following table:

CONDITION	Conditional
E	$IFE
NE	$IFNE
L	$IFL
NL	$IFNL
G	$IFG
NG	$IFNG

where VALUE is the value against which the return code is to be tested.

where ANSWER is true or false according to whether the answer given by the conditional should be true or false.

The conditionals can now be tested by calling CHECK. For example, consider the call

CHECK 0, E, 0, TRUE

As a result of this the first command in CHECK becomes

TRUE 0, E, 0

This calls the file TRUE which sets the return code to 0, then executes

$IFE 0

This should give 'true', which means that file MESSAGE is called with parameter OK

MESSAGE therefore prints out

RETURN CODE = 0, $IFE 0 OK

If the $IFE had erroneously given 'false' then the message

RETURN CODE = 0, $IFE 0 FAILED

would be listed.

4.3 PROGRAMMING EXAMPLES

4.3.1 Using I/O SVC 1

The following examples illustrate the general programming techniques for handling data transfers with SVC 1. These examples are device-independent and refer to LUs which could be assigned to direct access files. Examples of techniques specifically related to direct access files (primarily to illustrate uses of SVC 7), are given in Section 4.3.4.

The first example uses I/O and wait.

```
         SVC          1,PARBLK    I/O SUPERVISOR CALL
         LH           R0,STAT     GET STATUS IN GENERAL REG. ZERO
         BM           ERROR       BRANCH IF BIT Ø = 1 TO ERROR
         .                           ROUTINE
         .
         .

         ALIGN        ADC
PARBLK   DC           X'4806'     FUNCTION CODE AND LU
*        CALLS FOR READ ASCII AND WAIT ON LU6
STAT     DS           2           RESERVED FOR STATUS AND DEVICE NO.
         DC           A(START)    STARTING ADDRESS OF BUFFER
         DC           A(END)      ENDING ADDRESS OF BUFFER
         DAS          2           UNUSED FIELDS
         .
         .
         .
START    DS           N           START OF BUFFER,LENGTH=N
END      EQU          *-1         FINAL BUFFER ADDRESS
```

In the previous example, the program requests that the system read ASCII data into the specified buffer and suspend the task until the transfer is complete. An I/O call specifying wait 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 LU6 were assigned to the line printer; or if the device is unavailable, as would be the case if LU6 were assigned to an inoperative device. If either of these exceptions apply, the task resumes execution without I/O taking place. The second example uses ERROR thus:

```
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 statement to reserve a location for the status and device number. This is acceptable when using I/O and wait because the contents of this halfword is always changed by the system. Upon return to the program it contains zero if the transfer was successful or a negative value (Bit 0 = 1) and the device number truncated to 8 bits if the transfer failed.

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

```
        SVC     1,PARBLK1   FIRST I/O SUPERVISOR CALL FIRST BUFFE
LOOP    SVC     1,PARBLK2   SECOND SUPERVISOR CALL SECOND BUFFER
        LH      R0,STAT1    CHECK STATUS OF FIRST CALL
        BM      ERROR1      BRANCH ON ERROR STATUS
        BAL     Rx,PROC1    BRANCH AND LINK TO PROCESS FIRST BUFFER
        SVC     1,PARBLK1   REFILL FIRST BUFFER
        LH      R0,STAT2    CHECK STATUS OF SECOND CALL
        BM      ERROR2      BRANCH ON ERROR STATUS
        BAL     Rx,PROC2    BRANCH AND LINK PROCESS SECOND BUFFER
        B       LOOP        REFILL SECOND BUFFER
        .
        .
        .

        ALIGN   ADC
PARBLK1 DC      X'4002'     READ ASCII PROCEED LU2
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
        DAS     2           UNUSED FIELDS
PARBLK2 DC      X'4002'     READ ASCII PROCEED LU2
STAT2   D       2           RESERVED FOR STATUS AND DEVICE NO.
        DC      A(START2)   START OF SECOND BUFFER
        DC      A(END2)     END OF SECOND BUFFER
        DAS     2           UNUSED FIELDS
```

In this example, the program issues three SVCs that refer to two parameter blocks. All the calls request read ASCII and proceed on LU2. As soon as the data transfers for the first call is started, the task resumes execution at the second SVC. If at this time 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 task is again allowed to proceed. At this point, since the system was able to start a second transfer on LU2, 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 task 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.

The third example uses the unconditional proceed option:

```
        SVC   1,PARBLK1     I/O SUPERVISOR CALL
        BTC   15,ERR1       TRANSFER CANNOT BE STARTED
        .
        .                   UNRELATED PROCESSING
        .

CHECK   LH    R0,STAT       GET STATUS IN REGISTER ZERO
        BZ    COMPL         ZERO STATUS MEANS SUCCESSFUL TRANSFER
        BM    ERROR         ANALYZE ERROR STATUS
        SVC   1,PARBLK2     I/O SUPERVISOR CALL (WAIT)
        .
        .
        .

        ALIGN ADC
PARBLK1 DC    X'5208'       READ BINARY UNCON PROCEED LU8
STAT    DC    X'0100'       RESERVED FOR STATUS AND DEVICE NUMBER
        DC    A(START)      STARTING ADDRESS OF BUFFER
        DC    A(END)        ENDING ADDRESS OF BUFFER
        DAS   2             UNUSED FIELDS
PARBLK2 DC    X'0800'       WAIT ON TRANSFER
        DAS   4             UNUSED FIELDS
```

In this example, the program request binary read with unconditional proceed. With this type of call, the task is never put in the I/O wait state. It is left in the ready state and resumes execution. The Condition Code (CC) of its Program Status Word (PSW) indicates the status of the request. If upon return to the program, the Condition Code is zero, then the system was able to activate a driver in response to the

request. The transfer may or may not have been started. The device assigned to LU8 actually exists, and it was not busy at the time of the call. If the Condition Code has a value of X'F', then the transfer could not be started because the device was busy. The Branch on Condition instruction following the SVC call checks these conditions. If device is not busy, the program can continue with unrelated processing. The program must eventually check the status of the call to determine if the data was properly transferred. Note that in PARBLK1 the status was initially set to a positive value, (any positive value will do). If at the time of checking, this value is zero, then the transfer was successful. If it is negative then there was a failure. If it is still positive, the transfer has been started but is not complete. In the example, the program issues a wait request. This is not required. It could continue with other processing and return later to check the status again.

4.3.2 Using System Services SVC 2

The following example uses the UNPACK, LOG MESSAGE, and PAUSE SVC 2 calls to notify the operator of SVC 1 I/O error.

```
LOC       SVC     1,PARBLK1     I/O CALL
          LH      R0,STAT1      GET STATUS AND DEVICE NUMBER
          BM      ERROR         LOG ERROR MESSAGE
          .
          .
          .
ERROR     SVC     2,PARBLK2     UNPACK R0
          SVC     2,PARBLK3     LOG MESSAGE
          SVC     2,PARBLK4     PAUSE
          B       CONTIN        CONTINUE EXECUTION
          ALIGN   ADC
PARBLK1   DC      X'4008'       READ FROM LU8
STAT1     DS      2             STATUS AND DEVICE NUMBER
          DC      A(START)      BUFFER ADDRESS
          DC      A(END)
          DAS     2             UNUSED FIELDS
PARBLK2   DC      X'0006'       4-BYTE HEXADECIMAL UNPACK
          DC      A(DEST)       DESTINATION
          ALIGN   ADC
PARBLK3   DC      X'0007'       LOG DIRECT ASCII
          DC      14            MESSAGE SIZE
          DC      C'I/O         ERROR MESSAGE TEXT
DEST      DS      4             UNPACK BUFFER
          ALIGN   ADC
PARBLK4   DC      X'0001'       PAUSE CODE
```

The LOGged message in this example comes directly from the LOG Message parameter block. Also note that the UNPACK BUFFER follows the LOG Message text and is included in the printout to appear as:

I/O ERROR xxxx

where xxxx is the UNPACKed contents of General Register Zero containing error status and device number returned after an unsuccessful read operation. This message is then followed by the PAUSE message indicating that the program has suspended itself and is waiting action from the operator.

In many situations, a program may require varying amounts of storage from one moment to another. If such a program were to seize at once all the memory that it might ever need, it might

have difficulty opening files (since OS/32-ST allocates File Control Blocks from memory space above the user's allocation). In order to provide for an orderly memory management discipline, while allowing unused space to be efficiently used by the system, the Expand call can be used in conjunction with the Get Storage call as follows:

```
            ST      QTY,GETSTORE+4    STORE REQUEST QUANTITY
RETRY       SVC     2,GETSTORE        TRY TO GET STORAGE
            LR      ADRS,ADRS         DID WE GET IT?
            BNZ     OK                BRANCH IF YES
            SVC     2,EXPAND          NO-EXPAND BY 256 BYTES
            BNZ     MEMFULL           BRANCH IF NO MEMORY
            B       RETRY             EXPAND OK - NOW RETRY
*           ALIGN   ADC
GETSTORE    DB      Ø,2               GET STORAGE PARM BLOCK
            DC      H'ADRS'
            DC      F'Ø'              AMOUNT TO BE STORED HERE
EXPAND      DB      Ø,20              EXPAND ALLOCATION
            DC      H'1'              ONE BLOCK (256 BYTES)
```

This routine is entered with the requested number of bytes in register QTY. A Get Storage call is attempted. If it is successful, the routine exits with the storage address in register ADRS. If the Get Storage request is unsuccessful, an Expand call is made, to expand by another 256 bytes. If this call is unsuccessful, the routine exits to MEMFUL. If the expansion succeeds, the Get Storage call is retried.

The Release Storage call should be used in conjunction with Contract Allocation when and if the program wishes to return memory to the system.

4.3.3 Using Overlay SVC 5

An example program is shown following to illustrate how to handle overlays. This program assumes the program segments were linked in overlay structure in the same manner as explained in Section 6.2.4 and the two overlays MAINP.OV1 and MAINP.OV2 reside on Logical Unit 2 and 3 respectively.

MAIN PROGRAM

```
*          EXTRN SUB1,SUB2
MAIN       EQU               *
           .
           .
           .
           SVC               5,PARBLK1     LOAD MAINP.OV1
           LB                RØ, STAT1     LOAD STATUS BYTE
           LR                RØ, RØ        CHECK STATUS
           BNZ               ERHANDLR      IF NOT ZERO, ERROR
           BAL               15, SUB1      CALL OV1 VIA SUB1
           SVC               5,PARBLK2     LOAD MAINP.OV2
           LB                R0, STAT2     LOAD STATUS BYTE
           LR                R0, R0        CHECK STATUS
           BNZ               ERHANDLR      IF NOT ZERO, ERROR
           BAL               15, SUB2      CALL OV2 VIA SUB2
                             .
                             .
                             .
                             ALIGN ADC
           PARBLK1           DS  8
           STAT1             DB  X'00'
                             DB  X'04'     LOAD AFTER REWIND OPTION
                             DC  H'2'      LOGICAL UNIT 2
                             ALIGN ADC
           PARBLK2           DS  8
           STAT2             DB  X'00'
                             DB  X'04'     LOAD AFTER REWIND OPTION
                             DC  H'3'      LOGICAL UNIT 3
                             END
           *
           *                 OVERLAY1
                             ENTRY  SUB1
           SUB1              EQU  *
                               .
                               .
                               .
                             BR  15        GO BACK TO MAIN
                             END
           *
           *                 OVERLAY2
                             ENTRY  SUB2
           SUB2              EQU  *
                             BR  15        GO BACK TO MAIN
                             END
```

In the previous example, the first fetch overlay call causes the task to be suspended. The O.S. resident loader is activated to load MAINP.OV1 from Logical Unit 2 on the top of MAIN. If upon return to the program, the status is not zero, control is passed to the error handler routine. If the status is zero, the task knows MAINP.OV1 was loaded successfully and can issue a call to SUB1 which is an entry point of MAINP.OV1. After return from SUB1, a second SVC 5 is issued. The task is suspended again by the system and MAINP.OV2 is loaded from Logical Unit 3 on the top of the main program and hence overlays on MAINP.OV1. After the second call is processed, the task is again allowed to proceed. The status is checked to make sure the load operation is successfully completed. If the status is zero, SUB2 is called.

4.3.4 Using Disc Files With SVC 7 and SVC 1

In order to demonstrate the use of disc files with SVC 7 and SVC 1, the following program is presented.

Assume File A, which has been previously allocated by:

```
AL  DISC:FILEA,CO,100
```

is a 100-sector Contiguous file that contains 150, 80-byte records. The records were blocked by the user program with a blocking factor of three and there are no spanning records over the sector boundary. In addition, the records are followed by a file mark. We wish to copy File A to a Chained file, File B.

The following program writes File A to File B.

```
START     SVC     7,OPENA          ASSIGN FILE A
          LB      R0,OPSTATA       LOAD STATUS BYTE
          LR      R0,R0            CHECK STATUS
          BNZ     ERROR1           IF NOT ZERO, ERROR
          SVC     7,OPENB          ALLOCATE & ASSIGN FILE B
          LB      R0,OPSTATB       LOAD STATUS BYTE
          LR      R0,R0            CHECK STATUS
          BNZ     ERROR2           IF NOT ZERO, ERROR
GO READ   LA      R1, BUFFER       LOAD BUFFER ADDRESS
          SVC     1,PARBLKA        READ RECORD FROM FILE A
          LH      R0,IOSTATA       CHECK STATUS
          BZ      DEBLOCK          IF ZERO, NO ERROR/EOF
          THI     R0,X'0800'       CHECK EOF
          BNZ     DONE             YES, DONE
          B       ERROR3           ELSE ERROR
DEBLOCK   ST      R1,WRSAD         MOVE START ADRS TO PARBLK
          AHI     R1,79            COMPUTE BUFFER END ADDRESS
          ST      R1,WREAD         MOVE END ADRS TO PARM BLK
          SVC     1,IOSTATB        WRITE RECORD TO FILE B
          LH      R0,IOSTATB       CHECK I/O STATUS
          BM      ERROR4           IF MINUS, ERROR
          AIS     R1,1             BUMP BUFFER POINTER
          CLI     R1,BUFFER+240    HAVE ALL THREE LOGICAL RECORDS
                                   PROCESSED?
          BL      DEBLOCK          NO, CONTINUE DEBLOCKING
          B       GOREAD           ELSE GO TO READ NEXT REC
DONE      SVC     7,CLOSEA         CLOSE FILE A
          LB      R0,CLSTATA       LOAD STATUS BYTE
          LR      R0,R0            CHECK STATUS
          BNZ     ERROR5           IF NOT ZERO, ERROR
          SVC     7,CLOSEB         CLOSE FILE B
          LB      R0,CLSTATB       LOAD STATUS BYTE
          LR      R0,R0            CHECK STATUS
          BNZ     ERROR6           IF NOT ZERO, ERROR
                  .
                  .
                  .
          ALIGN   ADC
OPENA     DC      X'40A0'          ASSIGN SREW
OPSTATA   DB      0,1              TO LU1
          DS      4
          DC      C'DISKFILEA'
          DC      C'     '
          DS      4
          ALIGN   ADC
OPENB     DC      X'C0E1'          ALLOCATE AND ASSIGN ERW
OPSTATB   DB      0,2              TO LU 2
          DB      0,0              PROTECTION KEYS
          DC      H'80'            LOGICAL RECORD LENGTH
          DC      C'DISKFILEB'
          DC      C'     '
          DC      F'1'
```

```
            ALIGN   ADC
READA       DC      X'4801'         READ,WAIT,LU 1
IOSTATA     DB      0,0
            DC      BUFFER
            DC      BUFFER+255
            DS      8
            ALIGN   ADC
WRITEB      DC      X'2802'         WRITE,WAIT,LU 2
IOSTATB     DB      0,0
WRSAD       DC      BUFFER
WREAD       DC      BUFFER+79
            DS      8
BUFFER      DS      256
            ALIGN   ADC
CLOSEA      DC      X'0400'         CLOSE LU1
CLSTATA     DB      0,1
            DS      24
            ALIGN   ADC
CLOSEB      DC      X'0400'         CLOSE LU2
CLSTATB     DB      0,2
            DS      24
            .
            .
            .
            END
```

The task first assigns File A to Logical Unit 1 with access privilege SREW, thus preventing any other attempts to write on this file. Since File A is not protected, protection keys are not required. Upon return to the program, status is checked. The task next allocates File B to LU two with access privilege ERW, making File B unavailable to other assignments. Allocate and Assign are performed by one supervisor call instruction. It should be pointed out that for a Chained file, no disc space is actually allocated at assignment time and space is allocated only when it is needed. If there are no errors detected in assign, the task starts to read records from File A. Since File A was created with a blocking factor of 3, the task is

looping in the deblocking routine in order to pull the logical records out. When a logical record is pulled out, the task then writes this record onto File B sequentially. At the end, the task closes both files.

APPENDIX 1

SYSTEM COMMANDS AND MESSAGES

1. OPERATOR COMMAND

1.1 Notation

The syntactical rules given below are annotated in a modified Backus-Naur Form (BNF). The conventions of this notation are as follows:

1. Terminals. The actual characters and strings of the command language are called terminals and are composed of capital letters, numerals, and punctuation, except for the special characters < and > .

2. Nonterminals. The names of classes of terminals are composed of lower-case letters and are enclosed in angle-brackets, to distinguish them from terminals. For example, <verb> names a class of terminals. These class names are nonterminals.

3. Definitions. A nonterminal may be defined as being composed of groups of terminals, of nonterminals, or of groups of mixed terminals and nonterminals. A definition is set forth as follows:

<item being defined> ::= <definition>

The characters ::= mean 'is defined as'.

4. Selection. Whenever a choice may be made between alternative items in a definition, these alternatives are stacked one above the other and are enclosed in braces. For example:

<word> ::= { <noun> / <verb> / <adjective> }

5. Options. If an element of a definition is optional, it may be enclosed in square brackets. For example:

<noun phrase> ::= [<article>] <noun>

means a noun phrase is defined as a noun, optionally preceded by an article.

6. Mnemonics. When a mnemonic is presented as part of a definition, the required characters are underlined but the non-required characters are not. Thus:

APPEND

requires only the character A as an abbreviation.

7. Repetition. Any part of a definition that can be repeated one or more times is indicated by placing a subscript and superscript on the right of the enclosing brackets. These may be angle brackets, square brackets or braces. The subscript indicates the minimum permissible number of repetitions; the superscript indicates the maximum. Thus:

$$\langle \text{address} \rangle ::= \langle \text{hex digit} \rangle_{1}^{5}$$

means an address is composed of from one to five hexadecimal digits. If there is no subscript, the superscript indicates the minimum as well as the maximum. Thus:

$$\langle \text{three-digit number} \rangle ::= \langle \text{number} \rangle_{3}^{3}$$

$$\langle \text{three-digit number} \rangle ::= \langle \text{number} \rangle^{3}$$

are identical definitions, meaning a three-digit number is composed of three repetitions of <number> .

If the superscript is n, that means there is no upper limit on the number of repetitions of a construct. For example:

$$\langle \text{data string} \rangle ::= \langle \text{datum} \rangle \left[, \langle \text{datum} \rangle \right]_{0}^{n}$$

means <data string> is defined as <datum> followed by from none to any number of the pair comma, <datum> . Note that in this context no difference exists between square brackets and braces.

8. In the interest of being clear rather than rigorous, such obvious definitions as

$$\langle \text{letter} \rangle ::= \{ A \ldots Z \}$$

are used. Furthermore, such obvious nonterminals as <any ASCII character> are not defined.

1.2 Command Lines

The following rules define command lines.

$$\langle\text{command line}\rangle ::= \left\{\begin{matrix}\langle\text{terminator}\rangle \\ \langle\text{ordinary commands}\rangle\left[\langle\text{blank}\rangle_0^n\ ;\langle\text{terminator}\rangle\right]\end{matrix}\right\}$$

$$\langle\text{terminator}\rangle ::= \langle\text{blank}\rangle_0^n \left\{\begin{matrix}\langle\text{comment}\rangle \\ \langle\text{start command}\rangle \\ \langle\text{css call}\rangle \\ \langle\text{carriage return}\rangle\end{matrix}\right\}$$

$$\langle\text{ordinary commands}\rangle ::= \langle\text{ordinary command}\rangle\left[\langle\text{blank}\rangle_0^n;\langle\text{ordinary command}\rangle\right]_0^n$$

$$\langle\text{comment}\rangle ::= *\langle\text{any ascii character other than carriage return}\rangle_0^n$$

$$\langle\text{ordinary command}\rangle ::= \langle\text{blank}\rangle_0^n \left\{\begin{matrix}\langle\text{magnetic tape/cassette command}\rangle \\ \langle\text{general system command}\rangle \\ \langle\text{task-related command}\rangle \\ \langle\text{device and file control command}\rangle \\ \langle\text{css command}\rangle\end{matrix}\right\}$$

1.3 Task-Related Commands

<start command>::= START<blank>[<address>][,<args to program>]

<args to program>::=<any ASCII character other than carriage return>$_0^n$

<task-related command>::= { <options command> | <load command> | <expand command> | <pause command> | <continue command> | <cancel command> }

<options command>::= OPTIONS<blank><task option>[,<task option>]$_0^n$

<task option>::= { HALF | FULL | UT | ET | AFCONT | AFPAUSE }

<load command>::= LOAD<blank><fd>[,<impure bias>[,<pure bias>]]

<impure bias>::=<address>

<pure bias>::=<address>

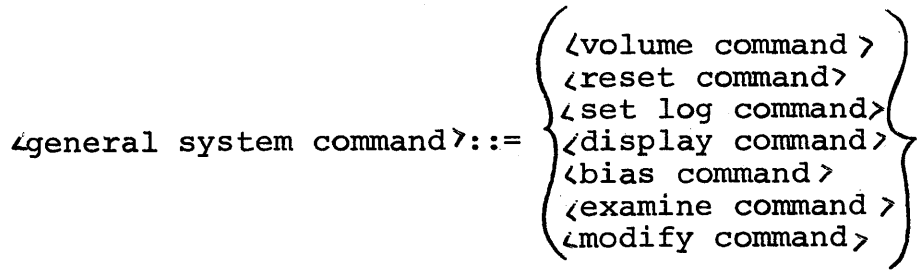
<expand command>::= EXPAND<blank><decimal number>

<pause command>::= PAUSE

<continue command>::= CONTINUE

<cancel command>::= CANCEL

1.4 General System Commands



⟨general system command⟩::= { ⟨volume command⟩ | ⟨reset command⟩ | ⟨set log command⟩ | ⟨display command⟩ | ⟨bias command⟩ | ⟨examine command⟩ | ⟨modify command⟩ }

⟨volume command⟩::= VOLUME⟨blank⟩⟨voln⟩

⟨reset command⟩::= RESET

⟨set log command⟩::= SET⟨blank⟩LOG⟨blank⟩[⟨fd⟩[,COPY]]

⟨display command⟩::= DISPLAY⟨blank⟩⟨display option⟩[,⟨fd⟩]

⟨display option⟩::= { LU | DEVICES | PARAMETERS | FILES,⟨voln⟩: { ⟨filename⟩ | - } . { ⟨ext⟩ | - } }

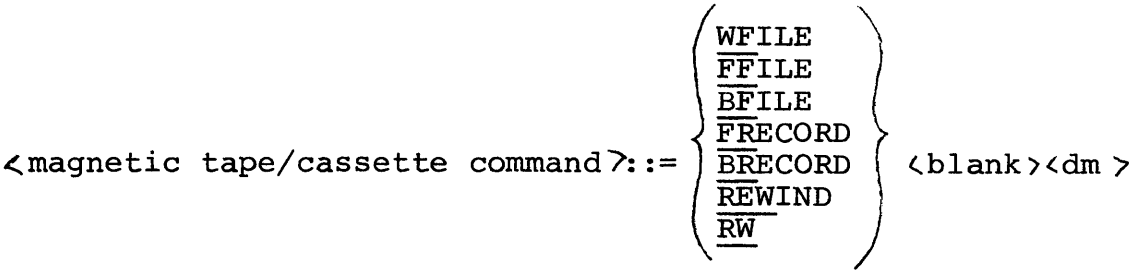
⟨bias command⟩::= BIAS⟨blank⟩⟨address⟩

⟨examine command⟩::= EXAMINE⟨blank⟩⟨address⟩,⟨decimal number⟩

⟨modify command⟩::= MODIFY⟨blank⟩⟨address⟩,⟨data⟩[⟨data⟩]$_0^n$

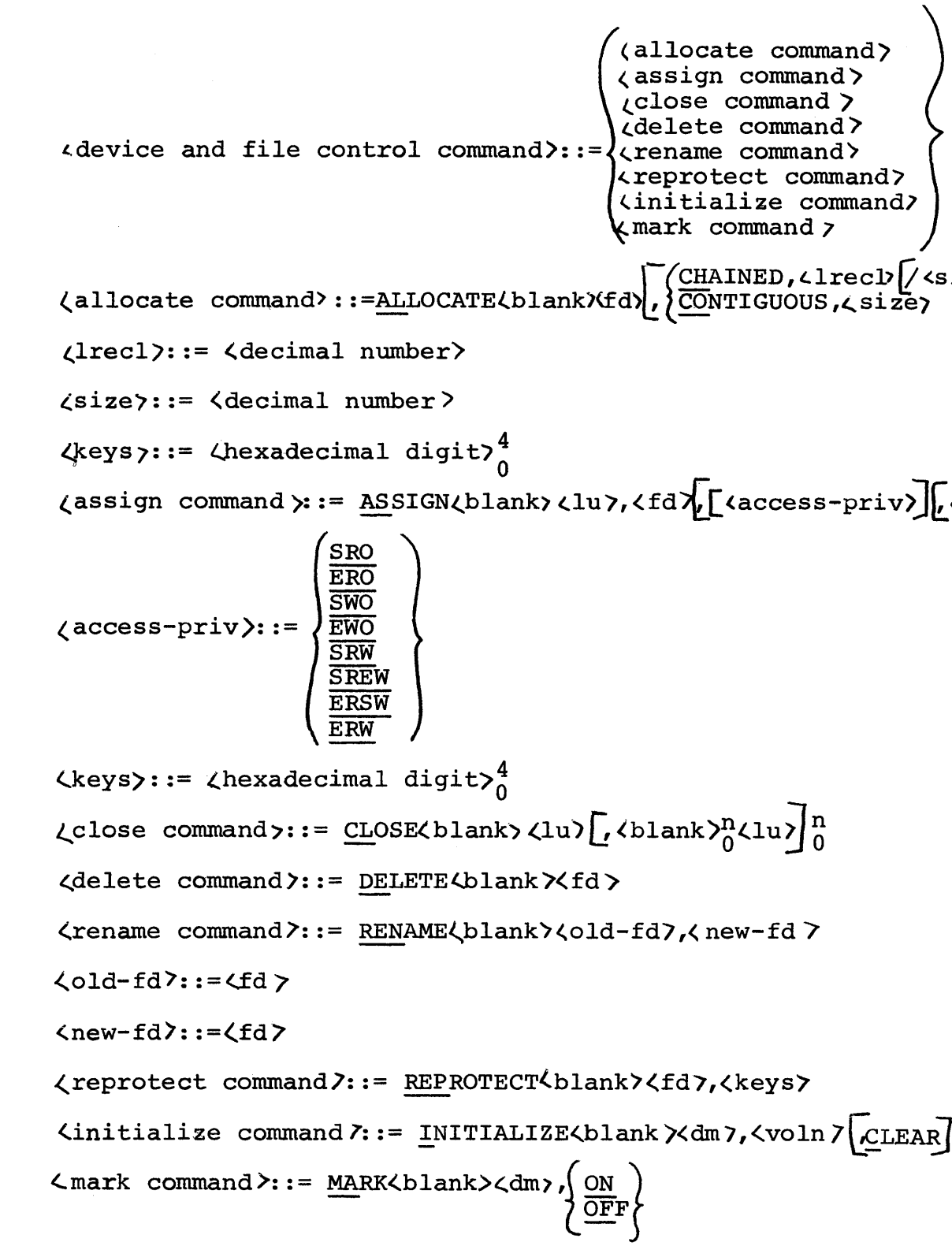
⟨data⟩::=⟨hexadecimal digit⟩$_0^4$

1.5 Magnetic Tape and Cassette Commands



⟨magnetic tape/cassette command⟩::= { WFILE | FFILE | BFILE | FRECORD | BRECORD | REWIND | RW } ⟨blank⟩⟨dm⟩

1.6 Device and File Control Commands



1.7 CSS Commands

⟨css call⟩::= ⟨fd⟩[⟨parameter list⟩]

⟨parameter list⟩::= ⟨parameter⟩ [,⟨parameter⟩]$_0^n$

⟨parameter⟩::=⟨blank⟩$_0^n$ ⟨any ASCII character other than carriage ret.⟩$_0^n$

⟨css command⟩::= { ⟨set code command⟩ | ⟨conditional command⟩ | ⟨file creation command⟩ | ⟨job control command⟩ | ⟨listing control command⟩ | ⟨exit command⟩ | ⟨clear command⟩ }

⟨set code command⟩::= SET CODE, ⟨decimal number⟩

⟨conditional command⟩::= { { $IFX / $IFNX } ⟨blank⟩⟨fd⟩ | { $IFE / $IFNE / $IFG / $IFNG / $IFL / $IFNL } ⟨blank⟩⟨decimal number⟩ | { $IFNULL / $IFNNULL } ⟨blank⟩$_0^n$⟨any ASCII char.⟩$_0^n$ | $ENDC }

⟨file creation command⟩::= { { BUILD / $BUILD } ⟨blank⟩⟨fd⟩ | ENDB | $ENDB }

⟨job control command⟩::= { $SKIP | $JOB | $TERMJOB }

⟨listing control command⟩::= { $NOCOPY | $COPY }

⟨exit command⟩::= $EXIT

⟨clear command⟩::= $CLEAR

1.8 Miscellaneous Nonterminals

<address>::= <hexadecimal digit> 5 0

<lu>::= <decimal number less than 255>

<voln>::= <letter> { <letter> / <digit> } 3 0

<dm>::= <letter> { <letter> / <digit> } 3 0

<fd>::= { [[<dm>] <voln> :] <filename> [. <ext>] }

<filename>::= <letter> { <letter> / <digit> } 7 0

<ext>::= { <letter> / <digit> } 3 0

<letter>::= {A...Z}

<digit>::= {0...9}

<hexadecimal digit>::= { <digit> / A...F }

(to be supplied)

2. PROGRAM COMMANDS

2.1 Supervisor Call

2.2 SVC 1

2.3 SVC 2

2.4 SVC 3

2.5 SVC 5

2.6 SVC 7

3. SYSTEM MESSAGES

4. ERROR MESSAGES

APPENDIX 2

OS COMPATIBILITY AND HALFWORD MODE

The question of software compatibility between OS/32-ST and other INTERDATA operating systems is multifaceted and must be discussed separately with regard to four different areas. These areas of compatibility are:

* SVC Compatibility
* File Compatibility
* Operator Command Compatibility
* Memory Management Compatibility

Section 1 below examines the compatibility issue in general. Section 2 is directed specifically towards OS/32-ST and explains executing programs under OS/32-ST in the halfword mode.

1. COMPATIBILITY AREAS

1.1 SVC Compatibility

OS/32-ST is capable of executing programs in either halfword or fullword mode. Because of OS/32-ST's dual nature, the subject of SVC Compatibility must consider the system to be two systems; a halfword OS/32-ST and a fullword OS/32-ST.

Table A2-1 shows the extent of SVC compatibility between BOSS, DOS, RTEX, OS/16-MT, RTOS, OS/32-ST (fullword), and OS/32-ST (halfword).

An entry in the table of Y refers to a Function supported in a fully compatible manner among all operating systems. An entry of N refers to a function not supported by that operating system. Some functions are supported in a slightly incompatible manner and these entries are so noted.

All SVC parameter blocks in the halfword mode under OS/32 are exactly compatible with the form of the same parameter block in all other operating systems. However, some SVC parameter blocks in OS/32-ST fullword mode due to the differing lengths of address constants and halfword/fullword boundary requirements, are not the same as in halfword mode OS/32-ST. Programmers may write parameter blocks in CAL common mode that can be translated appropriately into the fullword or halfword mode. The following restrictions must be observed:

- Every parameter block must be preceded by an ALIGN ADC statement
- All addresses within parameter blocks must be specified with the A(x) construction or the typeless construction.
- All halfwords within parameter blocks must be specified with the H'x' construction; fullwords must be specified with the F'x' or Y'x' construction.

Various SVC parameter blocks are illustrated in Table A2-2 along with the CAL common mode code that will produce these blocks for the appropriate target machine. The parameter blocks for the SVCs supported under OS/32-ST in halfword mode are given.

1.2 File Compatibility

File compatibility must be approached from two aspects, volume portability and program compatibility. File compatibility is illustrated in Table A2-3.

The BOSS/RTOS file structure is program compatible with the OS/32 ontiguous file. The DOS Direct-Access file structure is also compatible with ontiguous files. The Chained file is nearly, but not precisely, compatible with BOSS and RTOS, and is very nearly compatible with the DOS non-direct-access file. The primary differences are that the Chained file has a fixed logical record length (as in DOS), does not support filemark operations, and is fully open-ended.

There is a more overriding question however. When a file is built under one operating system, can it be read under another? This is the question of volume portability. The portability matrix for INTERDATA operating systems is also illustrated in Table A2-3.

1.3 Operator Command Compatibility

In general, the structure of operator commands is not compatible between all INTERDATA operating systems. In some cases this is necessary because of the differences in structure and capabilities of the operating systems themselves. For example, most commands in RTOS and OS/16-MT contain references to task ID's whereas those of BOSS, DOS, and OS/32 ST do not. References to physical devices are made by device address in BOSS and RTOS, by Logical Unit in

DOS, and by file name or device mnemonic in OS/32.

Table A2-4 shows operator command syntax for commands which are functionally similar (note each OS may have other commands not shown in the table).

1.4 Memory Management Compatibility

The question of memory management is best investigated by examining the table of constants returned to the calling task via SVC 2, code 5, Fetch Pointers. Table A2-5 shows this relationship. The constants table returned in OS/32-ST fullword mode contains fullword address entries; in the other operating systems the entries are halfword.

2. HALFWORD MODE

2.1 Hardware Architecture Considerations

The hardware architectures of the 16-bit and 32-bit machines differ as follows:

16-bit vs. 32-bit general registers
Single vs. double set of general registers
Altered lower memory allocation scheme
Altered channel control blocks
Static vs. volatile display panel status
Fullword allignment of data required.

Because of machine differences related to I/O and interrupts, compatibility is restricted to user programs. User programs must adhere to the following restrictions:

1. Do not use privileged instructions.
2. Do not reference memory beyond the program itself. This includes references to system data structures.
3. Use only legal SVC calls defined for the OS.

2.2 Program Conversion

To convert an OS/16 system to an OS/32 environment, it is necessary to:

1. Recode any special drivers used in the system.
2. Modify any programs in the system that use privileged instructions or illegal SVC calls (see Section 1.1 for SVC compatibility).
3. Preceed all SVC parameter blocks with ALIGH ADC statements and use address and fullword constructions as indicated in Section 1.1.
4. Assemble the program(s) using CAL with a target of 7/32.

2.3 Program Compatibility

Compatibility of User tasks can be achieved in two ways.

2.3.1 HW Mode

The Halfword Mode capability of the 7/32 Processor is supported under OS/32-ST. This mode is appropriate for 16-Bit tasks that are compatible with the features listed in Section 1. In this mode no Fullword Mode features of the OS can be used, and the task may not function on future INTERDATA Processors which do not support the Halfword Mode.

2.3.2 CAL Common

The CAL assembler provides a Common Mode which enables a source program to be assembled for either the 16-bit or 32-bit machines. This approach to compatibility involves adhering to the rules for CAL Common Mode plus the use of ALIGN ADC preceding all SVC parameter blocks. The CAL approach is appropriate for:

a. Expanding the scope or range of an OS/16 task to utilize more than 16 address bits.
b. Achieving compatibility for tasks on machines that may not have the halfword mode.

The most direct and straight-forward approach for guaranteeing program compatibility is:

a. Write all programs to be user programs (i.e., use no priviledged instructions) in CAL common mode.
b. Do not use special device drivers.
c. Follow the SVC rules in Section 1.1.

2.4 OS/32-ST SYSGEN Considerations

When the user configures a system using the configuration utility program (CUP), halfword mode support must be specified. Failure to do so will leave halfword mode support out, allowing only fullword mode programs.

2.5 Loading and Executing Halfword Mode Programs

Halfword mode programs are capable of addressing only 64KB of memory (e.g., from address X'0' to X'FFFF'). Prior to loading the program, the system must be placed in halfword mode. This is done with the OPTIONS operator command specifying halfword mode. Once

the system is in halfword mode, the ensuing LOAD commands activate the halfword loader. The loader will issue an error message if the program it is loading goes beyond the 64KB memory limit. The START command may now be used to start the program.

2.6 Subtle Differences

OS/32 conventions force a little different treatment of various features that are otherwise compatible in Halfword Mode.

2.6.1 SVC (Test and Set)

This compatible function on Contiguous direct access files returns a Condition Code of either zero or X'F' to the SVC 1 call. If the first halfword in the retrieved sector is neither zero or X'FFFF', a Condition Code of zero is returned by OS/32, but RTOS leaves the Condition Code unchanged.

2.6.2 SVC 2, Code 4 (Set Status)

OS/32 provides only one Arithmetic Fault interrupt; consequently, the Fixed Point and Floating Point options that are available on 16-bit architectures are combined in OS/32. That is, Options X'14' and X'04' both Enable Arithmetic Fault in halfword mode.

2.6.3 SVC 5 (Fetch Overlay)

The overlay name field in the SVC 5 parameter block is ignored in OS/32 ST. However, under OS/32 ST, the logical unit from which the overlay is loaded may be assigned to a named direct access file.

TABLE A2-1 SVC COMPATIBILITY

SVC NUMBER \ OS		BOSS	DOS	RTEX	OS/16MT	RTOS	(HALF) OS/32ST	(FULL) OS/32ST
1		Y	Y	Note 1	Y	Y	Y	Y
2	1 Code	Y	Y	N	Y	Note 2	Y	Y
	2	Y	Y	N	Y	Y	Y	Y
	3	Y	Y	N	Y	Y	Y	Y
	4	Y	Y	N	Y	Y	Y	Note 2
	5	Note 3	Note 3	N	Note 3	Note 3	Note 3	Note 3
	6	Y	Y	N	Y	Y	Y	Note 2
	7	Y	Y	Y	Y	Y	Y	Note 2
	8	N	N	Y	Y	Y, Note 2	N	N
	9	N	N	Y	Y	Y	N	N
	10	N	N	N	Y	Y	N	N
	11	N	N	N	Y	Y	N	N
	12	N	N	Y	Y	Y	N	N
	13	N	N	N	N	Y	N	N
	14	N	N	N	N	Y	N	N
	15	N	N	N	N	N	N	Y
	16	N	N	N	N	N	N	Y
	17	N	N	N	N	N	N	Y
	18	N	N	N	N	N	N	Y
	19	N	N	N	N	N	N	Y
	20	N	N	N	N	N	N	Y
	21	N	N	N	N	N	N	Y
3		Y	Y	Note 2	Y	Y	Y	Note 2
4		N	Y	N	N	N	N	N
5		N	Y	N	Note 2	Note 2	Y	Note 2
6		N	Note 4	Note 4	Note 4	Note 4	N	N
7		N	N	N	N	N	N	Y
8		N	N	N	N	Y	N	N
10		N	N	Y	Y	Y,Note 2	N	N
15		N	N	Y	N	N	N	N

NOTES:

1. SVC 1 I/O supported via IOSET2 module.
2. Additional options provided.
3. SVC is executed compatibly, but the constant table differs as described in the following discussion on Memory Allocation.
4. DOS SVC 6 totally incompatible; RTEX upward compatible to OS/16-MT; OS/16-MT interprets the RTOS proceed options and RTOS load option somewhat differently due to its being a memory only system.

Table A2-2 PARAMETER BLOCK COMPATIBILITY
(Halfword Mode OS/32ST)

SVC 1 I/O

Halfword Mode

	0 7	8 15
0	FC	LU
2	STATUS	PA
4	START ADDRESS	
6	END ADDRESS	
8	RANDOM ADRS.	
10	Generated but not used	

Fullword Mode

	0 7	8 15
0	FC	LU
2	STATUS	PA
4	START ADDRESS	
6		
8	END ADDRESS	
10		
12	RANDOM ADRS.	
14		
16	SIZE OF LAST	
18	XFER	

```
ALIGN ADC
DB FC,LU
DC H'0'
DC A(START)
DC A(END)
DC A(RANDOM)
DAS 1
```

SVC 2 CODE 1 Pause

	0 7	8 15
0	00	01

	0 7	8 15
0	00	01

```
ALIGN ADC
DB 0,1
```

SVC 2 CODE 2 Get Storage

	0 7	8 15
0	00	02
2	REGISTER #	
4	# BYTES	

	0 7	8 15
0	00	02
2	REGISTER #	
4	# OF BYTES	
6		

```
ALIGN ADC
DB 0,2
DC H'REG'
DC A(SIZE)
```

SVC 2 CODE 3 Release Storage

	0 7	8 15
0	00	03
2	# OF BYTES	

	0 7	8 15
0	00	03
2	fill	
4		
6	# OF BYTES	

```
ALIGN ADC
DB 0,3
DC H'REG'
DC A(SIZE)
```

SVC 2 CODE 4 Set Status

	0 7	8 15
0	00	04
2	NEW PSW STATUS	

	0 7	8 15
0	OPT	04
2	NEW PSW STATUS	

```
ALIGN ADC
DB OPT,4
DC X'xxxx'
```

SVC 2 CODE 5 Fetch Pointer

	0 7	8 15
0	00	05
2	REGISTER #	

	0 7	8 15
0	00	05
2	REGISTER #	

```
ALIGN ADC
DB 0,5
DC H'REG'
```

SVC 2 CODE 6 Unpack

	0 7	8 15
0	OPT	06
2	DEST ADDRESS	

	0 7	8 15
0	OPT	06
2	fill	
4		
6	ADDRESS	

```
ALIGN ADC
DB OPT,6

DC A(DEST)
```

PARAMETER BLOCK TABLE (Con't)

SVC 2 CODE 7 LOG MESSAGE

Halfword Mode

0	OPT	07
2	LENGTH	
4	TEXT	

Fullword Mode

0	OPT	07
2	LENGTH	
4	ADDRESS/TEXT	
6		

CAL Code

```
ALIGN ADC
DB OPT,7
DC H'LENGTH'
DC C'TEXT'
    OR
ALIGN ADC
DB OPT,7
DC H'LENGTH'
DC A(ADDRESS)
```

SVC 5 FETCH OVERLAY

Halfword Mode

0	OVER-	
2	LAY	
4	NAME	
6	00	OPT
8	LOGICAL UNIT	

Fullword Mode

0	OVER-	
2	LAY	
4	NAME	
6	fill	
8	00	OPT
12	LOGICAL UNIT	

CAL Code

```
ALIGN ADC
DCC'OVLYNM'
IFZ ADC-2
ORG *-2
ENDC
DB 0,OPT
DC H'LU'
```

TABLE A2-3. FILE COMPATIBILITY

File Structure

	BOSS	DOS	RTOS	OS/16MT	OS/32ST
Contiguous	Y	Y	Y	Y	Y
Chained	N	N	N	N	Y

Volume Portability

	Read Under				
Written Under	BOSS	DOS	RTOS	OS/16MT	OS/32ST
BOSS	Yes	No	Note 1	Note 1	No
DOS	Note 2	Yes	Note 2	Note 2	No
RTOS	Yes	No	Yes	Yes	No
OS/16MT	Yes	No	Yes	Yes	No
OS/32ST	Note 3	No	No	No	Yes

Note 1: A file written under BOSS can be read under RTOS or OS/16MT only if it is allocated on cylinder boundaries.

Note 2: A DOS file can be read under BOSS or RTOS only if it contains no overflow cylinders and if the user can find out where on the disc it is allocated.

Note 3: An OS/32 file can be read under BOSS only if it is a Contiguous file and if the user can find out where on the disc it is allocated. (Such a file could also be read under RTOS if it happened to begin on a cylinder boundary; however, such an assumption strains the limits of credibility).

TABLE A2-4. OPERATOR COMMAND SYNTAX COMPATIBILITY

FUNCTION	BOSS	DOS	RTEX	OS/16-MT	RTOS	OS/32-ST
Allocate file	ALxxxx,ssss,eeee	AL NNNNNN,lupa,ccc		Note 1	ALLO fnpa,ssss,eeee,wprp	AL fd,ft,size,keys
Activate file		AC NNNNNN,lupa				AS lu,fd,ap,keys
Initialize Pack	SA pa	PA pa,A				I dm,voln,options
Set file Attributes		AT NNNNNN,xxxx				
Delete file		DE NNNNNN		Note 1	RELE fnpa	DE fd
Close		CL				DL lu
List file	LE pa	LI lu [pa]			LIST pa	DI f,voln
Run program from file		RU NNNNNN [,pa]				
Assign logical unit	AS lu,pa	AS lupa[,lupa....]	ASSI TASKID,lu,pa		ASSI TASKID,lu,pa[,...lu,pa]	AS lu,fd,ap,keys
Set bias		BI xxxx	BIAS xxxx			
Load a program (task)	LO pa	LO lu			LOAD TASKID,pa	L fd,i,p
Start a program (task)		ST [xxxx]	STAR TASKID		STAR TASKID,pa,TADR,hhmmss	ST [xxxx]
Halt a program (task)	HA	@	HALT TASKID			P
Cancel a task			CANC TASKID			
Delete a task					DELE TASKID	
Continue a program (task)		CO	CONT TASKID			CO
Connect a program (task)					CONN TASKID pa,parm[,...pa,parm]	
Set task options					OPTI bbbb bbbb bbbb bbbb	O options
Display logical unit		LU			DISP TASKID	DI I
Pass message to task			TELL TASKID, message			
Set task's priority			PRIO TASKID,xx			
Set timeout count					TOUT TASKID,0xxx,xxxx	
Set date			DATE mm/dd/yy			
Set time			TIME hh:mm:ss		TIME hhmmss	
Read date			RDDA			
Read time			RDTI			
Open memory cell		OP xxxx	OPEN xxxx[,m]		OPEN xxxx [,xxxx...]	EX xxxx,n
Replace memory contents		RE xxxx	REPL xxxx,yyyy [,yyyy....]			MO xxxx
Open preceding cell						
Open succeeding cell						
Print system map					MAP	
Protect system					PROT	
Set TSKCOM size					TCOM xxxx	
Backspace record	BS pa				BKSP pa	BR fd
Backspace file mark	BF pa	BF lu			BSFM pa	BF fd
Forward space record	FS pa				FRSP pa	FR fd
Forward space file mark	FF pa	FF lu			FRFM pa	FF fd
Rewing device	RW pa	RW pa			REWI pa	REW fd or RW fd
Write file mark	WF pa	WF lu			WTFM pa	WF fd
Transfer command input	TR lu					
Copy file		CP lu,lu,[dddd,A]				
Position subfile	PO pa,NNNNNN	PO NNNNNN, lu				

TABLE A2-5 MEMORY MANAGEMENT COMPATIBILITY

OS / Pointer	BOSS	DOS	RTEX	RTOS	OS/16MT	(Half) OS/32ST	(Full) OS/32ST
CTOP	Y	Y	N	Note 1	Note 1	Note 1	,Note1
COMBOT	Y	Y	N	Y	Y	Y,Note2	Y,Note2
UTOP	Y	Y	N	Y	Y	Y	Y
UBOT	Y	Y	N	Y	Y	Y,Note3	Y,Note3
LDBIAS	Y	Y	N	N	N	Y	Y
XFRADR	Y	Y	N	N	N	Y	Y

Note 1 CTOP is the last halfword of physical memory in BOSS and DOS. It is top of program's allocated memory in RTOS, OS/16-MT and OS/32-ST. Top of memory in OS/32-ST is indicated by MTOP; bottom of system file control blocks and buffers (below MTOP) is indicated by FBOT.

Note 2 COMBOT is always set equal to CTOP in OS/32-ST.

Note 3 UBOT is set to the bottom of the program being loaded (always zero in RTOS), but in OS/32-ST UBOT is always set to the top of the operating system.

APPENDIX 3

GLOSSARY

Certain terms, which are used often in this document and which must be understood by the reader in order to gain a good understanding of the concepts presented are defined below.

ABSOLUTE - A term applied to an object module meaning that it must be loaded into memory at a specific location which cannot be altered at load time.

ACCESS METHOD - A well-defined technique for identifying, retrieving and storing specific records in a file. The access method is chosen at the time of a read or write call on the file.

ACCESS PRIVILEGE - An attribute of a system resource which defines a level of protection on that resource. Resources may be requested by a user for shared or exclusive access and specific I/O capabilities within that access. Examples of access privileges are: shared read only (SRO), exclusive read, shared write (ERSW).

ALLOCATION - The process of creating a file on a direct access volume.

ASCII (American Standard Code for Information Interchange) - The standard code used for information interchange among data processing communications systems. ASCII is a 7-bit code plus 1 bit which may be used for parity check. INTERDATA software uses

ASCII internally. The Assembler generates 7-bit ASCII with the parity bit always set to zero. Synonyms: USASCII, ANSCII.

ASSEMBLER - An assembler translates programs written in a symbolic machine language into a form which can be conveniently loaded into the system by a loader program. INTERDATA provides Assembler programs which number convert assembly language programs into binary object format.

ASSIGNMENT - The process of binding a specific physical device or direct access file to a Logical Unit.

ATTRIBUTES - The possible physical/logical functions and characteristics of any given device or direct access file.

BLOCK (OR PHYSICAL BLOCK) - A physical unit of data accessed by a single input/output operation.

BUFFER - An area of main memory used for the transfer of blocks to and from external storage. A buffer may be internal to the user program, or external to it, depending on the buffer management method.

BULK STORAGE - Storage of large-volume capacity used to supplement the Processor's internal memory. Discs (direct access devices) and magnetic tapes are used for bulk storage.

COMPILER - A compiler accepts programs expressed in a given language; i.e., the argument language, and produces corresponding programs expressed in a second language; i.e., the function language. For example, FORTRAN V produces as its function language the INTERDATA assembly language.

DIRECT ACCESS DEVICE - Any bulk storage medium that can be accessed in a random manner. Under OS/32 the smallest addressable unit is a sector.

FILE - A collection of related records, treated as a unit and organized for reference in a well-defined manner.

FILE CONTROL BLOCK (FCB) - A control block dynamically allocated as required whenever a direct-access file is assigned. FCB's contain information required for the processing of a file.

FILE DESCRIPTOR - An ASCII string of characters which abide by certain syntactial rules and shows the location of direct-access files or now direct-access devices.

FILE STRUCTURE - The physical means whereby a file is organized on a volume for reference by a buffer management method and access method. The file structure of any given file is fixed at the time the file is initially allocated.

FULLWORD - A collection of data, 32 bits (four bytes) in length, processed as a unit.

HALFWORD - A collection of data, 16 bits (two bytes) in length, processed as a unit.

LOAD MODULE - A module capable of being loaded into an operating system environment, and executed as a task.

OBJECT MODULE - The output of a language processor.

PRIVILEGED INSTRUCTIONS - Instructions that relate to I/O change the state of the Processor.

PROCESSOR STATUS - WORD (PSW) - The 64 bit PSW defines the state of the Processor at any given time.

PROGRAM - A sequence of statements possessing some implicit or explicit order of execution, and specifying a computer-oriented representation of a process.

PROTECTION KEYS - A set of values associated with a device file which must be supplied by users in order to obtain access to that device or file.

REAL-TIME - Capability of performing computations related to a physical process during the actual time that the process transpires. This is generally required where the results of the computation are to be used in guiding the physical process.

RECORD (OR LOGICAL RECORD) - A collection of data items treated logically as a unit.

RELOCATABLE - Term applied to an object module meaning that it may be loaded into memory at any location.

SECTOR - In general, the smallest segment of a direct access file which may be randomly addressed, specifically in OS/32, a 256-byte unit of storage.

SYSGEN - (System Generation) The building of a system from a library of source or object modules.

SYSTEM QUEUE - A feature of the Processor used by the operating system to aid in scheduling.

SOURCE MODULE - A mnemonic or easy to read representation of a program to be input to a language processor.

SUPERVISORY CALL (SVC) - The SVC Instruction is the means whereby the user program is enabled to communicate with the Operating System and to use the facilities thereof.

TASK - A general term for any program using an Operating System, as distinct from a system program which is considered part of the Operating System itself.

TASK CONTROL BLOCK - The interface between tasks and system code is provided in an internal System Table called the Task Control Block.

TASK OPTIONS - These variables, which the user may select, affect the mode of operation of his task.

TASK STATUS - The state of a task during its execution (e.g., Dormant).

VOLUME - A physical unit of external storage media, for example a disc cartridge or pack.

8-BIT LOADER - A loader which reads 8-bit bytes from some binary device such as a paper tape reader and stores the bytes directly into memory.

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.

STAPLE

STAPLE

FOLD

FOLD

FIRST CLASS
PERMIT No. 22
OCEANPORT , N.J.

BUSINESS REPLY MAIL

NO POSTAGE NECESSARY IF MAILED IN U. S. A.

POSTAGE WILL BE PAID BY:

INTERDATA®

2 Crescent Place, Oceanport, New Jersey 07757

TECH PUBLICATIONS DEPT. MS15

FOLD

FOLD

STAPLE

STAPLE