10-0000011-3

MONROE OPERATING SYSTEM
PROGRAMMER'S REFERENCE MANUAL

September 1981

MONROE SYSTEMS FOR BUSINESS
The American Road
Morris Plains, N.J. 07950

2109-S

The material contained herein is supplied without representation or warranty of any kind by Monroe Systems For Business. Monroe assumes no responsibility relative for the use of this material and shall have no liability, consequential or otherwise arising from the use of this material or any part thereof. Further, Monroe reserves the right to revise this publication and to make changes from time to time in the content hereof without obligation to notify any person of such revision or changes.

PURPOSE OF THIS DOCUMENT

This document is a Programmer's Reference Manual. It is to be used by experienced programmers as a reference tool. It is not intended for use as a learning aid by non-programmers.

RECORD OF CHANGES

Change No.	Date	Pages Affected	Description of Changes
1-2	6/81	All	Reviewer's Changes
-3	9/81	All	Preliminary Edition

TABLE OF CONTENTS

PART 1 - OPERATING SYSTEM MANAGEMENT

TABLE OF CONTENTS (Cont.)

TABLE OF CONTENTS (Cont.)

TABLE OF CONTENTS (Cont.)

TABLE OF CONTENTS (Cont.)

TABLE OF CONTENTS (Cont.)

TABLE OF CONTENTS (Cont.)

TABLE OF CONTENTS (Cont.)

TABLE OF CONTENTS (Cont.)

TABLE OF CONTENTS (Cont.)

TABLE OF CONTENTS (Cont.)

TABLE OF CONTENTS (Cont.)

TABLE OF CONTENTS (Cont.)

TABLE OF CONTENTS (Cont.)

TABLE OF CONTENTS (Cont.)

TABLE OF CONTENTS (Cont.)

LIST OF ILLUSTRATIONS

LIST OF TABLES

LIST OF TABLE (Cont.)

SECTION 1
INTRODUCTION

SECTION 1
INTRODUCTION

1.1 OVERVIEW

This manual is a description of the Monroe Operating System. It can be broken down into two major components: a software component, and a hardware component. The software component consists of the various Supervisor Calls (SVC's) which are assembly Language Programs that perform Operating System Services. The hardware component consists of the various operating system devices together with their related handlers and device drivers. These two components are then integrated through the System Executive into the Monroe Operating System Proper.

1.2 DOCUMENT CONTENTS

This manual is divided into an Introduction, Part I Operating System Management, Part II Input/Output Management, a series of appendices, an index, and a glossary. The two part division of the manual reflects the separation of the operating system into software and hardware modules.

Part 1 - Operating System Management

Operating System Management is concerned with the Monroe Operating System as it pertains to the preparation, management, and execution of tasks. The basic function performed by the Monroe Operating System Management is to integrate software and hardware modules through the System Executive and execute Supervisor Calls which allow the user to access the Operating System directly.

Part 2 - Input/Output Management

Input/Output Management is concerned with the Monroe Operating System as it pertains to the interaction of tasks with the various Input/Output devices. At the heart of the Monroe Input/Output Management is the console manager.

Appendices and Glossary

Abbreviations for the various device tables and control blocks are given in Appendix A. Appendix B contains the SVC error codes. Appendix C contains a listing of all of the SVC'stogether with

their functions, subfunctions, the bit patterns for their functions, and their use. The glossary is a quick reference for new or unfam- unfamiliar concepts.

1.3 HOW TO USE THIS MANUAL

The two parts of this manual can be grouped together into self contained sections which emphasize different aspects of the Monroe Operating System.

The first four sections (Sections 2 through 6) of Part I give a detailed overview of the operating system and is the logical place for the systems programmer to start. These sections outline the System Structure and Conventions, review the System Executive, indicate how Task Files are established, and show how the different parts of the operating system are integrated into a single functioning unit.

Sections 7 through 15 describe the various SVC's in numerical order. Each SVC is described in terms of its parameter block, parameter block functions, and a table which lists each SVC function together with all of the required fields necessary to implement that particular function.

Section 16 in Part II describes the Console Manager which allows the user to interface with the operating system directly. Since in many respects the Console Manager is independent of the Operating System Proper this material can be read without first consulting other sections in the manual.

Sections 17 and 18 describe the overall structure of the device drivers and the interrupt structures.

Sections 19 and 20 deal with the Data Structure and File Structures. Like the material on the SVC's these topics will be of particular interest to those programming at the Systems Programming Level.

1.4 RELATED MANUALS

Every attempt has been made to make this manual as self contained as possible. Nevertheless, there are many sections where an understanding of the material from other reference manuals would be helpful.

The material on Task Establishment can be read in conjunction with the TASK and ESTAB utilities in the MONROE UTILITY PROGRAMS PROGRAMMER'S REFERENCE MANUAL and the MONROE ASSEMBLY LANGUAGE PROGRAMMER'S REFERENCE MANUAL.

The material on the SVC's can be read in conjunction with both the MONROE ASSEMBLY LANGUAGE PROGRAMMER'S REFERENCE MANUAL and the Advanced Programming Section of the MONROE BASIC PROGRAMMER'S REFERENCE MANUAL.

Lastly, the material on the Console Manager can be read in conjunction with the Sections on the TASK and DEVICES utilities in the MONROE UTILITY PROGRAMS PROGRAMMER'S REFERENCE MANUAL.

PART I

OPERATING SYSTEM MANAGEMENT

SECTION 2
SYSTEM OVERVIEW

SECTION 2
SYSTEM OVERVIEW

2.1 INTRODUCTION

The Monroe Operating System is a Multi-Tasking Operating System. As such, it increases programming and operating efficiency yet requires a minimum of storage and computing time.

The Monroe Operating System is a disk-based operating system supporting 128KB of main memory. Up to 64KB is accessible by any one task. This 64KB space is called the logical address space which is divided into a 16KB system segment, a BLB pure code segment, and a 40KB data/code segment. These segments are mapped into the 128KB physical memory. (See Section 19.)

Facilities are provided for supporting up to 255 tasks running concurrently. Programs can reside permanently in memory or on a mass storage device, such as a disk, and be brought into memory for execution. Programs are executed on priority either through a hardware interrupt or via a programmed request. When a task of higher priority is finished, the operation of the lower task is resumed.

The operating system has a modular design, and is composed of two main module groupings corresponding to the hardware and software entrances to the Monroe Operating System proper. The hardware modules consist of all the device drivers, driver handlers, and driver interrupt handlers. The software modules consist of the various SVC functions and handlers.

The programmer communicates with the system through standardized requests and via commands from the system console. Input and output functions are carried out by driver programs, one for each type of device.

Since the operator plays an important role in the successful operation, it is necessary to understand the facilities available. These facilities and the operating procedures necessary to run the Operating System are described in this manual.

2.2 SYSTEM FUNCTION BLOCKS

The global function blocks of the Monroe Operating System are illustrated in Figure 2-1.

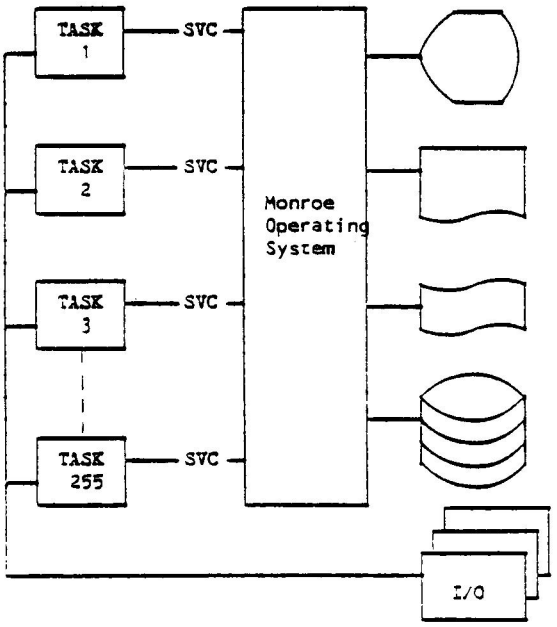


Figure 2-1. Monroe Operating System, System Block Diagram

The fundamental unit of work in the Monroe Operating System is called a "task". Tasks are described more thoroughly in Section 6 of this manual. For the time being, a task may be thought of as either a single program or as a main program together with a number of subroutines and overlays. The task interface to the Monroe Operating System proper is accomplished through SVC (supervisor call) instructions. Briefly, SVC instructions are assembly language programs which are executed by tasks which require operating system services. Most of the services provided by the Console Manager, for example, are performed by SVC instructions. These services then become available to user tasks. The parameters associated with each request are passed to an operating system parameter block.

The individual SVC instructions, together with their associated parameter blocks are described in Section 7 of this manual.

2.3 MONROE OPERATING SYSTEM FEATURES

The Monroe Operating System has the following features:

System Kernel

- Priority allocation of the whole computer system, including devices, execution time, memory allocation, etc.
- Dynamic memory handling that supports up to 128KB of main memory.
- Calendar and time-of-day are normally maintained by the system, and interval timing is available to user tasks.
- The number of tasks in memory at any time may be as great as 16, limited only by the amount of memory available and by system generation considerations.
- Tasks are scheduled by priority with 255 distinct levels. An optional time slice scheduler allows tasks of equal priority to share processor time.
- Low overhead. A minimum of searching is required to find the highest priority task.
- There are 16 system priority levels, 8 for hardware and 8 for software services. The system will support up to 255 interrupt driver devices. Response time for higher level interrupts is less than 200 us.

Input/Output Operations

- Input and output operations are device independent, allowing re-assignment without having to alter existing software.
- Devices and drivers can easily be added and deleted on line which allows the user to dynamically add new devices without a system generation.

Task Establishment

- Tasks and overlays are loaded into any free memory area up to 40KB in length by a relocating loader.
- Multiple applications programs can operate concurrently through the use of interleaving techniques.
- Tasks need not to be totally memory resident, but may be segmented and overlaid from any mass storage device under task control.
- Event-related information is maintained for each task within its own task queue.
- Tasks may request the activation and execution of other tasks, and may pass parameters to one another.
- Tasks may take traps on the reception of these parameters and upon completion of all types of requests.

File Management

- Comprehensive file management facilities are provided on direct-access devices; contiguous and indexed file structures are provided for safe and efficient use of all disks.
- Disk addresses are 32 bits.
- Hashed directory structures for faster directory searches.
- User readable directories.
- Twelve character file names plus file type designators.
- Cache memory type of sector buffering, to keep down the number of disk accesses.
- User File capabilities, keeps down the number of entries in the Master File Directory. User Directories speed up directory searching.
- User Files used for example in source module collection. The Module name is then expressed as a PROJECT.MODULE. Backup of User File Directory will backup all elements in the directory which simplifies bookkeeping of source modules since only one name is required.
- Byte random access, so a file may be treated as a stream of bytes.

- Variable or fixed length files.
- File access lockout in a multi user environment.
- Files can be accessed logical (byte access) or physical (256 bytes blocks).
- Files can be assigned by more than one program at a time unless the access attributes are violated.

2.4 SYSTEM START UP

The Monroe Operating System is normally loaded by a ROM storage device, the bootstrap loader. On completion of the load, control is normally transferred to the Console Manager Task which prints "MS8 Rx.yz" on the system console, where "x" is the release number and "yz" is the update number. The Console Manager then issues the command prompt "-". The system is now ready to accept commands. If the Monroe Operating System is generated without the Console Manager Task, control is transferred to a task specified by the user.

2.5 SYSTEM SHUT DOWN AND RESTART

In a disk-based system it is necessary that the system should be shut down or restarted in an orderly fashion, to assure the integrity of the disks in use. Before shutting down or restarting the system, the operator should cancel all non-system tasks, and close all disks. The system may now be restarted. If the system crashes, it should NOT be restarted. It must be reloaded.

2.6 SYSTEM CRASHES

When the system determines that further execution may cause system or user data to be destroyed, the operating system crashes in a controlled way. The operating system MUST be reloaded after a crash.

Figure 2-2 shows the principal interactions between the major groupings of the Operating System. For clarity, many minor interactions between these module groupings are not shown. Interaction between the different module groupings will be explained in subsequent sections.

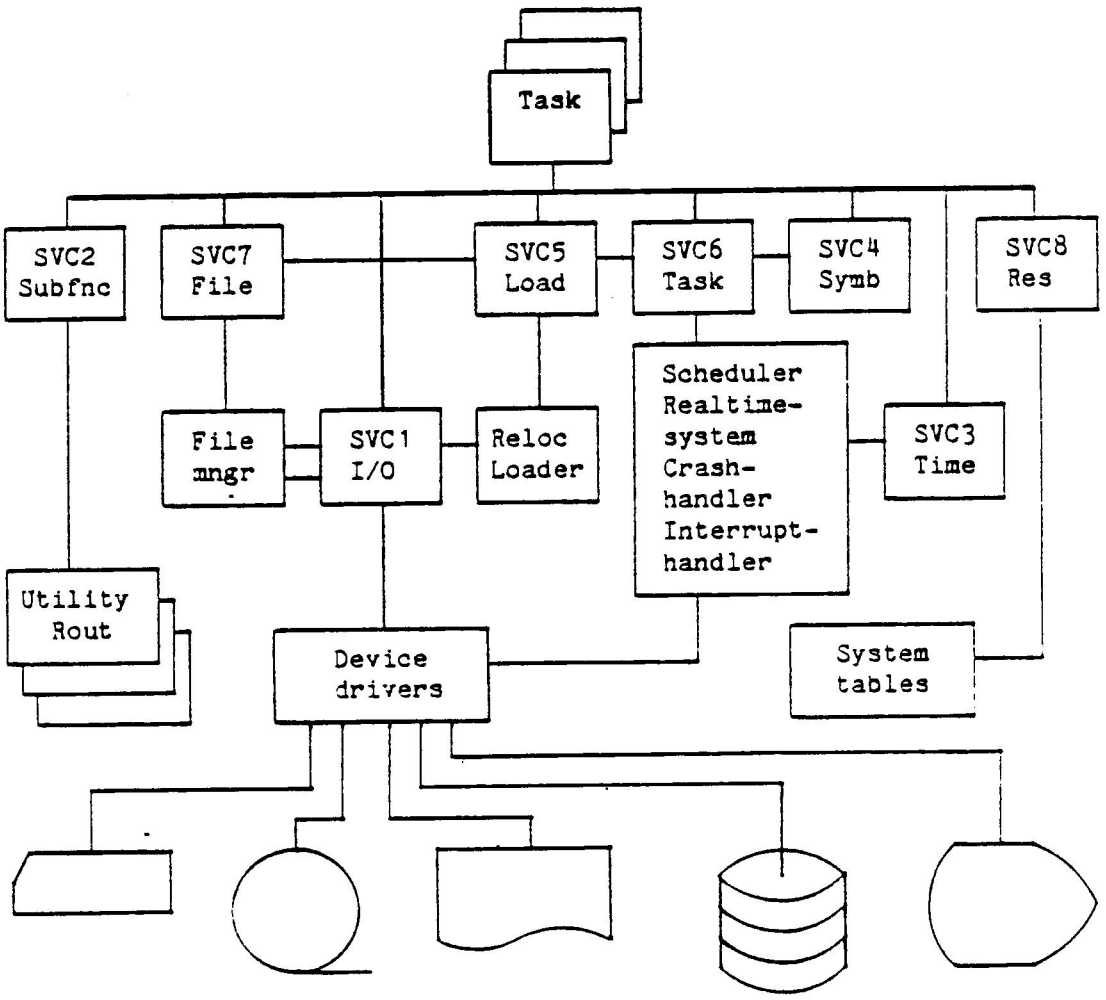


Figure 2-2 Monroe Operating System, Functional Block Diagram

SECTION 3
SYSTEM STRUCTURE

SECTION 3
SYSTEM STRUCTURE

3.1 INTRODUCTION

An outline of the system level structure of the Monroe Operating System is presented in this section. The Processor States, Interrupt System, and System Levels are described as well as how transitions between system levels are accomplished.

3.2 SYSTEM HIERARCHY

The Monroe operating system can execute programs in any of 16 system levels. That is, at any given instant a program will occupy one of these levels in the operating system. These levels are indicated by the following table:

Level	Description
0-7 (highest)	Hardware Drivers.
8	Software Drivers.
9	Queue Handling.
10	Real-Time Service.
11	System Queue Service.
12	Ready Queue Service.
13	Tasks (with 255 task priorities)
14	Reserved.
15 (lowest)	Idle Loop (Stop Mode)

Each of these levels is described in Section 3.6. Level 0 is the highest level and is reserved for one of the hardware drives (which occupy levels 0 through 7). Level 15 is the lowest level and is reserved for the Idle Loop (Stop Mode) of the operating system. Each of the 16 levels are currently used with the exception of level 14 which is reserved.

3.3 SYSTEM LEVELS

The levels indicated in Section 3.2 can be described as follows:

.Hardware Drivers, Level 0-7

These levels are reserved for the service of interrupt driven devices. The drivers can be triggered by either an external interrupt or a driver time out.

.Software Drivers, Level 8

This level is reserved for non-interrupt driven devices. The driver cannot be triggered by an external interrupt since it is not connected to an interrupt handler. Hence, it can be enabled only by a time out function.

.Queue Handling, Level 9

This level is entered when queues are scanned and modified by the system. For example, during real time processing events waiting to be executed must be held in queue before they are processed.

.Real-Time Service, Level 10

This level is reserved for update of all real-time dependent functions like the System Clock, etc.

.System Queue-Service, Level 11

This level is used to coordinate events controlled by the Resource control Block (RCB). Briefly, the Resource Control Block oversees the allocation of system resources. For example, to write to the printer you must coordinate the direct memory access processor, the printer controller, and the printer device. This is done for the RCB.

.Ready Queue-Service, Level 12

The Ready-Queue-Service handles all task dispatching and scheduling. It is also known as the Task Master. In particular, it schedules which programs should be run first.

.Tasks, Level 13

This is the system level where all tasks execute code. This level is divided into 256 priorities which are controlled by the Ready-Queue-Handler.

.Idle Loop (Stop Mode), Level 15

When no more code is to be executed, the system enters this level. The processor enters STOP MODE and all interrupts are enabled.

3.4 SYSTEM INTERRUPTS

As indicated above, the only way the system can go from a higher to a lower level is to execute an interrupt. This means there must be two types of devices supported by the operating system. Interrupt Driven Devices, and Non-interrupt Driven Devices.

An Interrupt Driven Device has an interrupt handler associated with it. A Non-interrupt Driven Device does not. Each interrupt driven device has an interrupt handler which can either enable or disable

the device during program execution. These are part of the interrupt system of the processor. In general, the interrupt system provides rapid responses to external or internal events that require service by special software routines. This is accomplished by means of an Interrupt Response Procedure. During the interrupt Response Procedure, the central processor preserves its current state while transferring control to the required interrupt handler.

3.5 SYSTEM STATUS

It was pointed out that the processor can preserve its current state while transferring control to some external device. This is done by means of the Program Status Word, PSW. Briefly, the PSW indicates the status of the current program running on the operating system.

The low nibble of the PSW contains the current system level (which as indicated can be any value from 0 to 15). For example, if the operating system crashes you can tell from the first four bits of the PSW what level (0 up to 15) the operating system was in at the time of the crash.

The high nibble contains some control flags such as the User-System-Stack Flag. Note changes in transition can occur at various times during processing and for various reasons. When an external event at a higher level then the one currently in operation causes a transition, the current PSW is incremented to a level above the previous one. When an external event at a lower level then the one currently in operation causes a transition the current PSW is decremented to a level below the previous one.

3.6 SYSTEM STATES

At any given instant the Central Processor can be in either Stop Mode or Run Mode. The transition from stop to run requires the occurrence of an interrupt. Since the current value of the PSW controls the operation of the system, as the contents of the PSW are changed, this enables the interrupt handlers for the various device drivers which

in turn enable the drivers themselves.

During the cold start a number of system tables must be initialized. In particular, initial values must be supplied to the Initial Value Table (IVT) which lsts the number of devices and drivers supported by the operating system. Also, various program modes must be established. This information is supplied by the operating system when it is loaded.

SECTION 4
SYSTEM CONVENTIONS

SECTION 4
SYSTEM CONVENTIONS

4.1 INTRODUCTION

The Monroe Operating System allows the user to run programs, tasks, or routines in any one of four well defined modes. They are, in increasing order of priority and privilege:

- .User Mode (UM)
- .System Mode User (SMU)
- .System Mode System (SMS)
- .Interrupt Mode (IM)

These modes are differentiated by a combination of PSW bits and status bits in either the Resource Control Block (RCB), Device Control Block (DCB), or Task Control Block (TCB) (c.f. the section on Data Structures Part II). These are the only modes of operation permissible on the Monroe Operating System. At any given instant the central processor will be executing code in one of these four modes. In addition each mode is defined by the following characteristics:

- .The system level at which the mode is operating.
- .The stack pointer and registers used by the program, task, or routine being executed in that mode.
- .The type of code being executed.
- .The task-ID and priority used by the operating system in executing the code.

4.2 USER MODE - UM

User Mode (UM) is the mode in which all user tasks run. Internal and external interrupts are enabled during execution. The only way in which you can exit from this mode is either by means of an interrupt or by executing a supervisor call (SVC). The system level for the user mode is the task level and the type of code it executes is user code.

4.3 SYSTEM MODE USER - SMU

System Mode User (SMU) is the mode in which system code is executed on behalf of a task. The code is organized, scheduled, and dispatched as though it were a routine of the task. The SMU mode

is entered at a supervisor call, therefore all higher system levels are enabled. The system level for the SMU mode is the task level and it only executes system code.

4.4 SYSTEM MODE SYSTEM - SMS

This mode occurs when the system changes critical information such as queues. It is non-reentrant, and the Ready Queue Service Interrupts are disabled at the time of entry. While the system is in this mode no new task can be dispatched, hence any routines which run in this mode should be short and quickly executed. This mode is exited via an external interrupt.

The primary difference between the SMS and SMU modes of operation is in the system level for SMS. SMS executes system code at a higher level then the task level. Ideally, this should seldom occur. For example, when you are using an SVC-8 function to add an item to a linked list, you are in SMS, even though it is a task being executed. In this case the operating system would be executing on a queue level which is higher than the task level.

4.5 INTERRUPT MODE - IM

This mode is used only for interrupt service routines within the Device Drivers, Real-Time Update, System Queue Service, Ready Queue Service, and Idle Loop. All higher system levels are enabled. The system level for the IM mode is the task level and it only executes system code.

4.6 FURTHER CONVENTIONS

With regard to the above system modes. The following conventions are in effect:

.REGISTER USE

For definition, the Primary Register Set includes both registers AF, BC, DE, HL, Y, X and CS, IL. The Secondary Register Set includes AF', BC', DE' and HL'.

The operating system never uses the secondary register set without first saving them temporarily on the stack and then restoring them when it is finished. Both register sets are stored on the task's stack when the task is not running.

.SUBROUTINE CONVENTIONS

Parameters are passed in registers or in memory such as the System Pointer Table (SPT), Task Control Block (TCB), etc. Register modification within a subroutine is normally done according to the function of the subroutine. All other registers are normally preserved. Exits to unlabeled addresses are not permitted.

.SVC FUNCTION CONVENTIONS

All SVC instructions cause the system to enter SMU state. The address of the parameter block is passed in register Y on entry to the SVC handler. It is the responsibility of the SVC handler to check the validity of any parameters passed in the parameter block.

.INTERRUPT CONVENTIONS

Interrupts cause control to be passed to the individual interrupt handler in the IM state. The address of the control block is passed in register X to the interrupt handler.

SECTION 5
EXECUTIVE DESCRIPTION

SECTION 5
EXECUTIVE DESCRIPTION

5.1 INTRODUCTION

There are two ways to enter the Monroe Operating System: through a software service request (SVC) or through a hardware interrupt. This section contains a description of the request modules that accomplish these functions.

5.2 EXECUTIVE MODULES

The Monroe Operating System can respond to a software or hardware request. The modules associated with each type of request and their interconnection are illustrated in Figure 5-1.

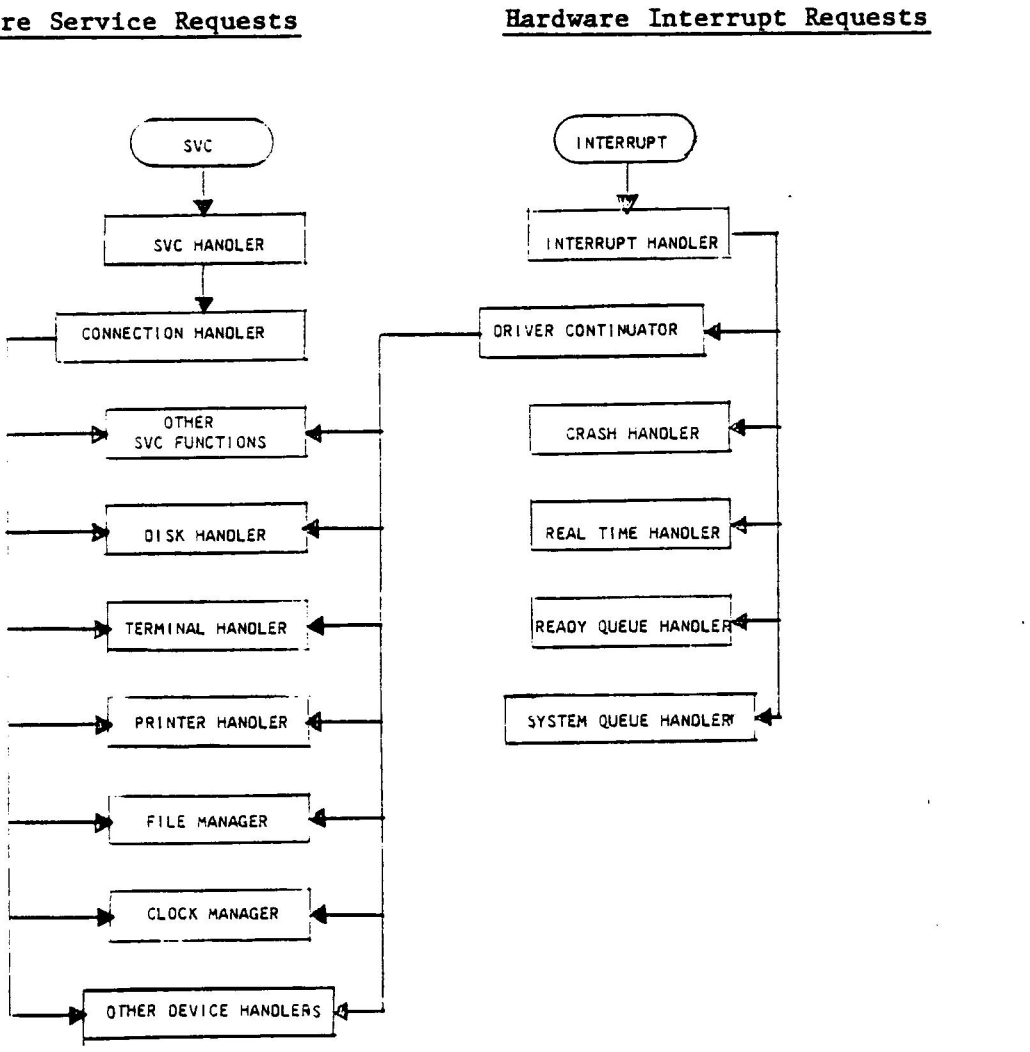


Figure 5-1. Monroe Operating System, Software and Hardware Modules

All executive routines act as a subroutine of the calling task and work in the state refer to Section 4.3. The register usage by the software modules are indicated below:

Register	Use
X	Control Block
A	Return Status
Carry (Flags)	0-Function not complete
	1-Function complete

Executive routines cannot issue SVC calls.

5.3 SYSTEM INITIALIZATION

The Monroe Operating System is initialized at the system start up. On entry into the operating system the PSW (Program Status Word) is not known. Therefore the first operation that is performed is to put the processor into a privileged uninterruptable state. This is done through the following sequence of steps:

1. The System Pointer Table (SPT) is cleared.
2. The SPT is given initial values from data in the Initial Value Table (IVT).
3. The dynamic data structures (Part II) are then created using an SVC8.
4. The rest of memory is scanned to build a dynamic memory pool.
5. The first task is then started through the SVC-Handler and the operating system tries to enter an Idle Loop.

5.4 SYSTEM RESOURCES

A resource (as used through this manual), is a task or an area of memory that can be used by a task. Since tasks (for example programs) can use other tasks (for example utility commands) every task can be treated as a resource. There are two types of resources:

1. Shared Resources
2. Exclusive Resources

These resources are coordinated through the use of resource control blocks.

1) Shared Resources

A shared resource is any resource that can be used by several tasks at the same time. For example, a common area of memory, or any reentrant task is á shared resource.

2) Exclusive Resources

An exclusive resource is one which can be used by only one task at a time. Any task which wants to access an exclusive resource must wait in a queue. The resource is given to the first task in a queue when the resource is free. For example, device drivers, memory allocaters, non-reentrant code are all examples of exclusive resources.

5.5 RESOURCE TYPE

Most requests to a resource have to wait in a queue before they are processed. All queuing is done on a priority basis. Within a given priority resources are allocated on the basis of first come, first serve. Obviously operating system resources must be controlled and coordinated somehow. This is the purpose of the Resource Control Block (RCB).

The MONROE Operating System contains three major types of resources:

1. Volumes
2. Devices
3. Tasks

Each of the above groups has two reference roots; one for symbolic handling and one for numeric handling.

1) Volumes

A volume is a directory oriented mass storage device. Volumes can be used by tasks to copy from, to write to, and in general to store data on.

2) Devices

A device can be a real physical device (like a disk drive, tape drive, or controller). Or it can be an imaginary device (like the NULL device which will accept any input and do nothing with it).

3) Tasks

A task is any program which can be executed.

5.6 RESOURCE CONTROL BLOCK (RCB)

Coordination of operating system resources is accomplished by the Resource Control Block. The Resource Control Block in turn is

managed by the connection and disconnection handler through a series of subroutines. The Resource Control Block can be thought of as a list structure. This list structure is generated by the SVC8-Handler.

For example, suppose you want to control access to the disk. Since the disk can be used by a task it is a resource. However, in order to control the disk you must first gain access to the direct memory access processor, the disk controller, and the disk device. These three resources must therefore be coordinated by the RCB as if they were a single resource.

This is accomplished by subroutines which connect to, queue to, disconnect from, and release entries in the RCB list. In general, a task will not be connected to any particular RCB until such time as it can be connected to all required RCB's needed for its execution to prevent deadlock conditions.

5.7 SVC HANDLER

In order to enter the operating system through the software door an SVC instruction must be executed. All SVC instructions cause entry into the SVC handler. The function of the SVC Handler is to perform preprocessing which includes saving the primary register set of the indicated instruction, verifying the rest size of the stack, and so on.

The SVC Handler also looks for the specific SVC number of the instruction being executed. If it finds it, it calls the Connection Handler to connect to the system resource or service the SVC instruction it is requesting. Some SVC's such as SVC2 have second level handlers that perform similar preprocessing.

5.8 SVC FUNCTIONS

There are a number of SVC functions within the operating system that are initiated by SVC instructions. These functions then become available to tasks. Each SVC has a handler which is normally a fully reentrant resource. In addition, any new SVC Handler may be added to the operating system either at cold start or during run time.

The following SVC handlers are all exclusive resources within the operating system itself:

.SVC8
.SVC2.1
.SVC2.10
.SVC2.11

SVC 8

Is used for resource manipulation, for example, to add resources to the operating system or remove resources from the operating system.

SVC2.1

Is used to allocate memory or deallocate memory. For example, allocating memory to the terminal monitor.

SVC2.10

Is used as a Bit Map Manager within a Volume. For example, this SVC can be used by the file management system.

SVC2.11

Is used as a Directory Manager within a volume. This SVC can also be used by the file management system.

5.9 FILE MANAGER

The File Manager includes all the logic necessary to support the Monroe File Management System. The File Manager is called from either the SVC1 or SVC7 Executor. It then decodes each function specified by the parameter block and invokes the necessary executors. At completion each Executor determines if any other request is still outstanding. Control is returned with an appropriate status report when all functions have been processed or if the Executor encounters an error. Each Executor makes use of the following subroutines contained within the File Manager:

.Directory Management Subroutines
.Bit Map Management Subroutines

Directory Management Subroutines maintain information on all currently allocated files. Bit Map Management subroutines provide a method for allocating and deleting files on direct-access-volumes. The File Manager also contains SVC1 intercept routines which intercept all I/O calls into a file.

5.10 RESOURCE CONTROL BLOCK HANDLERS

The Resource Control block Handler controls the allocation of resources by the Operating System. Each handler is described below.

Connection Handler

The connection handler either enters a shared resource, or queues the request to an exclusive resource. When connected, the connection handler initiates the resource control block and calls the resource handler specified in the resource control block. Upon completion the connection handler calls the disconnection handler. If not completed, the connection handler either puts the task in completion wait, or returns to the SVC handler, depending on the function code in the parameter block.

Disconnection Handler

The disconnection handler is called to release a task from a RCB and its tree. When released, the next request for any path in the tree is propagated, and that task is removed from the connection wait state.

Termination Handler

When a task is disconnected from a RCB, an optional terminator handler is called. This terminator is called in the SMS/SMU modes, and performs some post-processing for that resource. This function is normally used by device drivers to do work that should not be done within the interrupt handler.

Non-maskable Interrupt Handler

This is a user defined handler, normally to handle power down and power up sequences. If not needed, a dummy should be included at system generation time.

Clock Interrupt Handler

This is a dedicated interrupt service handler, and is entered at a frequency defined by the hardware, normally 100 HZ (10 ms). This routine decrements the head of the interval queue, which is sorted by the time value. If the item elapses, the Real-Time Handler is triggered and an overflow counter will be started. All the following interrupts are now being counted in the overflow counter until the real-time service is done. Upon carry from the overflow counter and system overload, the Crash Handler is called.

Crash Handler

Checks are made for normally impossible states of the operating system. When such a condition is found the system brings itself to a halt before further destroying the conditions that led up to the impossible situation. This is done by the Crash Handler.

The Crash Handler saves all the CPU-registers in a crash diagnostic area. Then a programmed reset of the I/O system is initiated and a PSW is loaded that puts the system in an uninterruptable state.

Interrupt Handler

Interrupts are normally handled in a standard way. The program counter is stored on the current memory stack by the hardware. The primary register set is also stored on the stack. The system stack is then selected, if not already present, and the previous pointer is stored on this stack. The new system level is then selected.

The Interrupt Service Tables, corresponding to this new level are scanned to find the device from which the interrupt was generated. When it is found, a call is done to the specified interrupt service handler. The interrupt service handler then processes the interrupt. If the request is completed, the RCB will be added to the System Queue and the System Queue Handler is triggered.

The rest of the interrupt linkage is scanned to look for another interrupt from a device. If so, that continuator will be called. The next active device with a higher level is found and the interrupted process is restored.

If no device was found, the Illegal Interrupt Counter is decremented, and if it becomes zero the Crash Handler is called.

Real Time Handler

The Real Time Handler is triggered by the clock interrupt handler after the first item in the internal queue elapses. When the Real Time Handler is entered the first item in the internal queue is then removed. This item is examined to find out the reason for the handler's being triggered. Depending upon the reason one of the following actions will be updated:

Interval and time of day request
Time of day clock
Day and year calendar including leap year.
Driver timeout.

Device Drivers

Each type of peripheral or task device has a control program driver. These are normally fully reentrant, with the exception of the interrupt handling phase on dedicated drivers. New devices with drivers may be added to the system in the same way as SVC functions.

The initiation phase of a driver runs as a subroutine of the task in SMU state. The interrupt handling phase normally runs with all higher levels enabled in the IM state. The termination phase of a driver runs in a reentrant state although no task may be executing more than one termination phase subroutine at a time. Each device is controlled by a Device Control Block (DCB).

Ready Queue Handler

All tasks which are currently in ready state are in a queue called the Ready Queue, which is a linked list of the TCB. A task is dispatched for execution when it is at the head of the ready queue. If time slicing is disabled, the task currently executing remains in this state until it voluntarily relinquishes control or until a higher priority task is ready. If time slicing is enabled, the task relinquishes control when its time expires or if an equal priority task is ready. Therefore, if no equal priority task is ready, the task continues to execute for another time slice.

The Ready Queue Handler is triggered by:

- .The Real-Time Handler when a task becomes ready to execute after a real-time request, or when a task has exceeded its limit.
- .The System Queue Handler when a task is scheduled to execute the initiation or termination phase of an exclusive resource.
- .A task which lowers its priority below another ready task.

System Queue Handler

The system contains a System Queue which is a linked list of the RCB's. Whenever an item is added to this queue, an internal interrupt is initiated to schedule events coordinated by the RCB. The handler removes each item in the queue, and calls the Disconnection Handler.

The System Queue Handler is triggered by the System Interrupt Handler when a request to an interrupt driver device is complete.

System Pointer Table

The SPT contains necessary information for proper operation, and is used by the system with direct address instructions. Most the data in the SPT are pointers and roots. The System Stack, Interrupt Vectors and Interrupt Service Tables are also allocated in the SPT.

SECTION 6
TASK ESTABLISHMENT

SECTION 6
TASK ESTABLISHMENT

6.1 INTRODUCTION

The fundamental unit of work in the Monroe Operating System is the task. A task can be a single program or it may consist of a main program together with a number of subroutines and overlays. Tasks can either permanently reside in memory or they can be loaded into memory as required.

Every task is given a task identification which is four letters long when loaded. The number of tasks that can be loaded into memory at any given instant is specified at system generation.

Each task is controlled by the operating system through a task control block (TCB).

6.2 PREPARATION

Only task files can be executed. Object modules are prepared using the Task Establisher program ESTAB. The result is a relocatable load module of the task file. Once a task has been established it can be loaded into memory by the resident loader via the LOAD command (c.f UTILITY PROGRAM PROGRAMMER'S REFERENCE MANUAL) or by executing an SVC 6.

6.3 STATUS

Once a task has been loaded into memory it can be in any of five states. These are:

State	Abbreviation
1. Current	-
2. Ready	R
3. Waiting	W
4. Paused	P
5. Dormant	Dorm

1) Current

Is the state of the task that is currently executing instructions. Only one task can be in this state at any given time. All other tasks in memory are in one of the four other states and may become the current task depending upon circumstances.

2) Ready (R)

Is the state of any task that is ready to become the current task. It is eligible to be dispatched (i.e., become Current) whenever it has the status of the highest priority Ready Task.

3) Waiting (W)

Is the state of any task waiting for an event. A task is in the Wait State if it cannot become Ready until some specific circumstance has occurred. These are:

- .Connection Wait - Waiting for I/O to start.
- .I/O Wait - Waiting for I/O completion.
- .Time-Wait - Waiting for an interval or time of day.
- .Trap Wait - Waiting for a task-handled event.
- .Task Wait - Waiting to be released by another task.

4) Paused (P)

Is the state of any task that has been paused during execution. A paused task cannot continue to execute unless it is continued by either the console operator or by another task.

5) Dormant (DORM)

Is the state of any task that has not been started. A Dormant task is one which may not execute until it has been explicitly started, either by the console operator or by another task. When a resident task goes to EOT it enters the Dormant state. When any task is loaded, it enters the Dormant state after load-complete, and remains in this state until started. In addition, tasks can be of two types: Resident, Nonresident.

6.4 TASK TERMINATION STATUS

The termination status of a task can be any of the following:

Description	Abbreviation
1. Resident	R
2. Non-resident	(blank)

1. Resident
   Any task which is not deleted from the system after it completes execution.

2. Nonresident
   Any task which is deleted from the system after it completes execution.

Note: Task termination status is set at task establishment and can be changed until it terminates.

6.5 PRIORITY AND SCHEDULING

The Monroe Operating System recognizes 255 priority levels from a high of 1 to a low of 255. Of these levels, 1-255 are available to user tasks while 0 is reserved for system's use. Each task has two priorities associated with it:

1. Task Priority
2. Dispatch Priority

1) Task Priority

Task priority is the priority currently assigned to the task. It is set at task establishment time and may be modified by an operator command or an SVC-6.

2) Dispatch Priority

Dispatch priority is the priority set up by the system to determine the order in which ready tasks are serviced. Normally, a task's dispatch priority is the same as its task priority, but may be raised temporarily if the task is using a system resource required by a higher priority task.

6.6 TASK SCHEDULING

Two types of scheduling algorithms are available:

1. Tasks may be scheduled in strict priority.
2. Tasks may be time sliced with either a global slice limit or an internal slice limit relative to a given priority.

1) Strict Priority Scheduling

In the case of strict priority scheduling, if two tasks of equal priority are started, a task remains active until it relinquishes control of the processor. Care should be taken in assigning priorities so that tasks which do not frequently relinquish control of the processor do not inadvertently lock out other tasks. A task may relinquish control in one of the following ways:

1) It is Paused or Cancelled by the operator or by another task.
2) A higher priority task becomes ready because of some external event.
3) It executes an SVC that places it in a Wait, Pause or Dormant state.

2) Time Slice Scheduling

Rather than scheduling on a strict priority basis, tasks may be time-sliced within a given priority. This option allows the user to ensure that tasks of equal priority receive equal shares of processor time. The time-slicing option may be enabled and disabled by an operator command (c.f. MONROE UTILITY PROGRAMS PROGRAMMER'S REFERENCE MANUAL). When a task becomes ready, it is queued on a round-robin basis behind all ready tasks of equal priority.

6.7 SUPERVISOR CALLS (SVC's)

The program interface to the operating system is provided through Supervisor Call (SVC) instructions. SVC instructions are executed by programs in order to request operating system services. The PARAMETERS associated with the request are passed to the Monroe Operating System in a parameter block. For example, most of the services provided by the Console Manager are performed with SVC instructions, thus making these services available to user tasks. Section 7 gives a description of the individual SVC instructions and their associated parameter blocks.

6.8 EVENT QUEUE

The Monroe Operating System provides a facility at the task level known as the task-trap facility. This allows a task to handle asynchronous events. Event related information is maintained for each task within its own event queue, where an event queue is just a linked list of nodes.

A trap service occurs whenever the task is in a Trap-Wait state. If a task is in any Wait state other than Trap Wait, the trap does not actually occur until the task has left that Wait state. Several trap-causing conditions may occur before the first trap is handled by a task. Therefore, the event queue facility allows for queuing of event information during periods when the task is unable to service a trap.

The following trap-causing conditions cause an item to be added to the event queue:

1. Addition of a parameter to the event queue.
2. Completion of any queued no-wait request.
3. Requests to your own task-device.
4. SVC 1 requests to your own TCB.

6.9 TASK DEVICES

The Monroe Operating System recognizes special "imaginary" devices, Task Devices, that may be added to the operating system. These

devices have all the characteristics of a real interrupt driven device, since they can be assigned to any task and can have I/O requests queued for them. Whenever requests are queued another task is "activated" instead of a device driver (the normal procedure). The task "initiates" the operation and handles the completion of the request just as a device driver would do. Normally, the task device operates at a task priority level higher than any task that may call it.

A task device has more freedom than a device driver since it can execute SVC instructions. The task device simulates a real device driver and it can be written in such a way that it processes requests of a single task device, or of several similar or unrelated devices.

Task devices may be used to simulate devices, to spool devices (impose intermediate disk buffering), or to impose special formatting, communication protocols, or data conversions on existing interrupt driven devices. These techniques can be implemented in such a way that they are "transparent" to the task which uses them.

SECTION 7
SUPERVISOR CALLS

SECTION 7
SUPERVISOR CALLS

7.1 INTRODUCTION

Supervisor Calls (SVC's) are used in programs to request the Monroe Operating System to perform operating system services. These services include operations such as data transfer, file handling, task manipulation, timer coordination, device control, and subfunctions like text processing.

All SVC's are connected to a parameter block where each parameter is specified to perform some aspect of the requested function. The parameter block must reside in a writable segment of memory.

SVC's are written in assembly language and they always consist of a restart 7 instruction OFF HEX followed by the SVC request number (1 byte) and the SVC parameter (2 bytes). The argument specifies the request type. The SVC parameter is normally an address to a parameter block in a tasks B-segment (upper logical 40KB area). However, parameter values lower than 256 mean that one of the values in the CPU-register pair should be used (HL, BC, DR, DE).

All CPU-registers are saved on the task's stack when an SVC instruction is executed. During an SVC the system operates as a subroutine of the calling task, mostly within the callers priority. All CPU-registers, except the A-register and the Flags, are restored from the stack. The A-register contains the return status. An "error" is indicated by a true zero flag and a false carry flag while "no error" is indicated by a false zero flag and a true carry flag.

When an illegal SVC is entered, a message is logged on the system console and the task is paused. The task can then be cancelled or resumed (if a correction is made).

If the task has specified system recovery, a message is logged and the task is paused for each abnormal return status. When the task is told to continue, the SVC that produced the error is repeated. When system recovery has been specified, all error handling is done by a system dependent routine inside the operating system.

7.2 SVC CALLING CONVENTIONS

SVC's can be executed from MONROE BASIC, PASCAL, or Assembly Language programs. In all cases, the programmer supplies a parameter bock and the SVC types to the appropriate procedure within the language. The format of the parameter block is covered in the next section. The calling coinventions from Monroe Assembly Language, BASIC and PASCAL are described in subsequent paragraphs.

Assembly Language SVC Calling Conventions

There are two forms of SVC calls for assembly language programs: one which uses a register pointer (BC, DE, or HL) to the parameter block and another with a fixed pointer. The forms are:

1. SVC <type>, <register pointer>
2. SVC <type>, <address>

The <type> must be a constant byte value which indicates the SVC type. This type also indicates the format of the parameter block. The parameter block is indexed by an absolute address, <address>, or via a <register pointer> of BC, DE, or HL.

Note that register A and F have the result code.

Examples: Ex. 1 SVC 2,LOGGIN

EX. 2 SVC 7, S7DEVASG

Ex. 3 LA HL, S7DEVASG
SVC 7,(HL)

NOTE: Examples 2 and 3 perform the same function but do not result in the same object code.

Monroe BASIC SVC Calling Conventions

BASIC has a single built in procedure of the form:

```
SVC <type>, <parameter>
```

where <type> is an integer expression which is the SVC type and the <parameter> is the name of a Monroe BASIC array which contains the parameter block. The procedure returns if no errors occur with the result code in the parameter block.

Example:

```
DIM  SVC2(10)          ! Parameter Block Allocation
SVCNUMBER = 2          ! SVC Type
<BASIC CODE>           ! Set Up Parameter Block
SVC SVCNUMBER, SVC2    ! Call SVC
```

Monroe PASCAL SVC Calling Conventions

Monroe PASCAL has a single built in Boolean function of the form:

```
FUNCTION SVC  (SVCTYPE: INTEGER
               ;SVC PARAMETER BLOCK:PACKEDRECORD)
               :BOOLEAN:
```

where SVCTYPE is the type of SVC being called and the SVC parameter block is a packed record whose size is a function of the SVC type. No type check is done between the record size and the SVC type. A sample record description would be:

```
BYTE = 0 ..255;
       PARAMETER BLOCK =
         PACKED RECORD
             FUNC   : BYTE  ;
             RESULT : BYTE  ;
             ETC    : INTEGER;
       END
```

The function returns a true value if the SVC was executed and a false value otherwise. The program will be paused if an illegal SVC error occurs.

7.3 THE PARAMETER BLOCK

The Parameter Block of an SVC contains all of the detailed information necessary to perform the function that is requested. All Parameter Blocks, irrespective of the SVC in question, must contain the function code field and the return status field. These fields are common to all Parameter Blocks. The contents and size of any remaining fields depend upon the service that is being requested. Parameter Blocks are divided into subblocks which describe the different parameters of the SVC. A description of this portion of the parameter block is covered in Section 8.

The function code field (S0.FC) is one byte and is the first byte in the parameter block. The resultant code S0.RS is also one byte and immediately follows the function code. All other bytes in the parameter block follow these two.

The description here and in Section 8 uses the assembly language naming convention to describe indexes and values. This convention uses two part names separated by a period (ex. FIRST.SECOND). The first part refers to a common item and the second refers to a particular index or value.

The SVC's extend this convention such that all SVC parameter block indices have the form S<n>.<field> (ex., S0.FC) where <n> is the SVC type and <field> is a field within the parameter block. Values within a field are designated by S<n>. F<name> where <name> is a descriptive mnemonic for the value.

This convention is recommended for assembly language programming and the names can be supplied from system files. Note that this convention is only applicable to assembly language programs.

7.4 FUNCTION CODE FORMAT

The function code byte (SO.FC) consists of three fields:

Bits	Description
0-5	SVC function (is type dependent)
6	Wait/Proceed
7	Unconditional Proceed

NOTE: 0 is the least significant bit.

The function type is a value from 0 to 63 and is a function of the SVC type.

The Wait/Proceed bit determines if the SVC is to be executed in an asyncronous manner. It can have the following values:

Value	Name	Description
0		Task will wait until SVC is complete
1	SOF.NW	SVC is initiated and the task continues. The task can determine that the SVC is done by checking the result code.

Wait/Proceed, SOF.NW

A wait call requests the operating system to suspend the calling task until completion of the requested operation.

Once a request has been initiated (that is, the specified resource is free), any proceed request causes control to be returned to the task so that the task may execute concurrently with the transfer. The return status is not set until completion of the request, except for an illegal function, and illegal resource which are rejected before initiation. Every no-wait request to a queued resource will be added to the event queue of the task, if enabled. The return status of the request may be checked by:

.Monitoring the return status field in the parameter block.
.Issuing a wait for completion request to the same resource.
.Taking a task handled trap on completion.
.No/Wait SVC calls must assign FF HEX to the result field if the result is to be polled by a program.

The uncondition proceed is used to determine if a resource is free. The bit can have the following values:

Value	Name	Description
0		Wait until the resource is free and then perform SVC.
1	SOF.PRO	Perform SVC if the resource is free, otherwise, return.

Unconditional Proceed, SOF.PRO

Unconditional proceed is used when a task does not wish to wait for the requested operation. Requests are coordinated by the system so that only one request may access an exclusive resource at a time. If an unconditional proceed is not requested and the specified resource is in use at the time of the request, the calling task is suspended by the operating system until the resource is free. At that time, the request is initiated.

7.5 RESULT CODE FORMAT

The result code consists of one byte that indicates the return status of an SVC. The list of error codes appears in Appendix B. The following is a list of some common codes from Appendix B.

Return Status	Status Code	Meaning
SOS.OK	0	No Error
SOS.EON	1	End of node in the task
SOS.IFC	2	Invalid function
SOS.PRO	3	Can't connect to the resource
SOS.OFFL	4	Resource off line
SOS.PRES	5	Not yet present in this system
SOS.NYET	6	Function not yet implemented
SOS.CAN	7	Request is cancelled
SOS.SVC	8	Invalid SVC function
	9	(Reserved)

If an unconditional proceed is specified and the resource is in use, the request is rejected. Return status is set to "3" and the condition code is set to nonzero. The calling task may then retry the request at a later time. If the specified resource is not in use, the setting of an unconditional proceed has no effect on the request.

7.6 SVC CONVENTIONS

Each SVC is discussed in terms of its Parameter Block and the Operation, Type, and particular functions defined by the arrangement of bits in its function code. Exercise to reconstruct what the bit-patterns are for each SVC from the tables. For the sake of completeness, the bit patterns for the Function Code, Return Status, and all other relevant parameters are given in Appendix C for each SVC.

SECTION 8
SVC 1 INPUT/OUTPUT REQUEST

SECTION 8
SVC 1 INPUT/OUTPUT REQUEST

8.1 INTRODUCTION

SVC 1 is used by a task to perform all general purpose I/O requests.

8.2 PARAMETER BLOCK

The contents of the parameter block for SVC 1 is shown below.

(0) S0.FC	(1) S0.RS	(2) S1.LU	(3) S1.TS
Function code	Return status	Logical unit	Term. status
(4) S1.BAD		(6) S1.BSZ	
Buffer start address		Buffer size in bytes	
(8) S1.BCNT		(10) S1.RND	
Byte count at completion		Random address -	

The parameter block for SVC 1 has the following structure:

	Offset	Bytes	Type	Mnemonic	Name
1)	0	1	Byte	S0.FC	Function Code
2)	1	1	Byte	S0.RS	Return Status
3)	2	1	Byte	S1.LU	Logical Unit
4)	3	1	Byte	S1.TS	Termination Status
5)	4	2	Address	S1.BAD	Buffer Address
6)	6	2	Integer	S1.BSZ	Buffer Size (Bytes)
7)	8	2	Integer	S1.BCNT	Byte Count
8)	10	2	Address	S1.RND	Random Address
	Total	12			

8.3 PARAMETERS

The parameter that appear in the parameter block are discussed below.

1) SO.FC, Function Code

The function code byte allocation is as follows:

Bit	Description
0-4	Operation
5	SVC 1 type:
	A) Read/Write Operation
	B) Special Operation
6	Wait-Proceed
7	Unconditional-Proceed

As seen above, bit 5 specifies the type of operation. Each is described below.

A) Read/Write Operation (For SVC 1 type Field = 0.)

The various functions in the operations field are determined by the decimal values of the contents of bits 0-4. These are summarized below.

Bit	Value	Name	Description
0-1			Read/Write
	0	SOF.WAIT	Wait for SVC Completion
	1	S1F.READ	Read
	2	S1F.WRIT	Write
	3	S1F.WRD	Write with read check
2-3			Data Type
	0	S1F.IASC	Image ASCII
	1	S1F.FASC	Format ASCII
	2	S1F.IBIN	Image Binary
	3	S1F.SPEC	Special
4			Access Type
	0		Sequential Record Access
	1	S1F.RND	Random Record Access via the random address field.

S0F.WAIT, Wait For Completion: A wait for completion request (Function Code 0) causes the task to be placed into a wait state until the completion of a previous request to the specified resource. If no such outstanding request exists, control is returned to the task immediately. For an explanation of READ, WRIT, and WRD see section 8.4.

S1F.IASC, Image ASCII: This is ASCII data without space compress and no termination character in the buffer.

S1F.FASC, Format ASCII: This is ASCII data with space compress. The most significant bit in the byte is set together with a seven bit space counter. The termination byte in the buffer is zero.

S1F.IBIN, Image Binary: This specifies the transfer of eight-bit data bytes without any formatting.

S1F.SPEC, Special: This parameter depends upon the drivers.

B) Special Operations

For SVC1 Type Field = 1 The various functions in the operationg field are determined by the decimal values of the contents of bits 0-4. These are summarized below.

Bit	Value	Name	Description
0-4	-	-	Special Operation
	0	S0F.TST	Test Request (see below)
	1	S0F.CAN	Cancel Request (see below)
	2	S1F.FR	Forward Record
	3	S1F.FF	Forward File
	4	S1F.WF	Write File Mark
	5	S1F.BR	Back Record
	6	S1F.BF	Back File
	7	S1F.RW	Rewind
	8	S1F.ATTN	Attention
	9	S1F.FEOF	Fetch EOF Position

SOF.TST, Test Request: A test request (Function Code 20H) returns with a return status of 0 if there is no outstanding proceed request to the specified resource by the task. If there is an outstanding proceed request, the call returns a status of FF.

SOF.CAN, Cancel Request: A cancel command request (Function Code 21H) is used to terminate a request which has previously been issued. This is especially useful on an interactive device. If the cancel request is not initiated; then any outstanding requests must first be completed before others can be started on the resource.

When a cancel command is issued, the operating system schedules the previous request for termination. The actual termination is asynchronous to the cancel request. When the request is terminated, if the task is enabled, it receives a trap. The parameter added to the trap queue is the address of the original parameter block, not the address of the cancel parameter block. Alternatively, the task may sense the completion of the request with test request or wait for completion request.

It is possible that a previous proceed request has gone to completion, at the time the cancel call is done, without being serviced, i.e. it has been added to the event queue. In that case, the return status in the proceed request will not contain the cancel return code.

Two parameter blocks are involved in the cancel processing. The first is the original parameter block specified by the user when the request was initiated. The second is the command function parameter block which is requesting the cancel. These should not be the same parameter block. At the completion of the cancel command, status is returned to both of these parameter blocks as indicated below:

1) Cancel parameter block:
   - 000 The requested termination has been scheduled.
   - 007 No request on-going for the task on this resource.
   - 011 Resource not assigned.

2) Original parameter block:
   007 Request is canceled.

SVC 1 Type: The SVC 1 type is bit 5 in the function code. Its value, 0 or 1, defines the type of operation executed by the SVC.

Wait-Proceed: Wait-Proceed is a common SVC attribute and is invariant with respect to SVC type. Wait-Proceed has the same bit-pattern structure for each SVC parameter block which is described in detail in Section 7.

Unconditional Proceed: Unconditional proceed is a common SVC attribute and is invariant with respect to SVC type. Unconditional proceed has the same bit-pattern structure for each SVC parameter block which is described in detail in Section 7.

2) S0.RS, Return Status

The Return Status codes will take on a value from 0 to 19 where 0-9 are standard error codes and 11-19 are the SVC 1 error codes listed below.

Function	Code	Meaning
	0-9	Common codes.
S1S.LU	10	Illegal or unassigned LU.
S1S.AM	11	Access mode missmatch.
S1S.TOUT	12	Time-out.
S1S.DWN	13	Device off line.
S1S.EOF	14	End of file.
S1S.EOM	15	End of media.
S1S.RER	16	Recoverable or parity error.
S1S.UNR	17	Unrecoverable, read-write failed.
S1S.RND	18	Invalid random address.
S1S.NRND	19	Non-existent random address.

3) S1.LU, Logical Unit

In order to provide device independent I/O, all I/O requests are directed to a logical unit. The logical unit in the parameter block is a user defined integer from 0 to 200 (decimal) which is assigned to one logical file or device. Logical units greater than 200 are used by the Operating System (SVC) and support programs like the debugger. The particular resource desired must be assigned to a specific logical unit prior to executing an SVC 1 call. If an invalid or unassigned LU is specified, the call is rejected unless reference to the system device numbers is allowed. If no operation is desired, the specified LU should be assigned to the NULL device.

4) S1.TS, Termination Status

This status byte may contain information unique to the specific type of device.

5) S1.BAD, Buffer Address

The buffer is specified by the buffer start address, and points to the first byte in the buffer. All buffers must be fully contained in the same logical segment of the task address space. Buffers used in read requests must be in a writable segment, since the memory locations are changed by the read operation.

6) S1.BSZ, Buffer Size

The buffer size specifies the number of bytes to be written, and the maximum number of bytes to receive.

7) S1.BCNT, Byte Count

This field is used to return the actual number of bytes transferred during a request. It is most useful when dealing with variable length record devices, such as magnetic tape. Byte count has an undefined error status.

8) S1.RND, Random Address

Random Address is used when the function code specifies random (S1F.RND). It is interpreted in two different ways depending on the format specified in function code.

On Image ASCII/Binary, it specifies the logical record number (starting at 0) to be accessed during data transfer.

On Formatted ASCII, it is divided into two 16-bit fields which contain positioning information at data transfer. These fields are input in the read-function and output in the write-function. The first field (most significant part) contains vertical tabulation and the second horizontal. Positioning is absolute, when the most significant bit is set, the remaining 15-bits specify an absolute position. Positioning is relative. When the most significant bit is off, the remaining 15-bits form a signed integer for relative positioning. For example, if we want the data to start on the first position of the next line, then contents of S1.RND in hex is 00018000.

Ex. 1

Implement a Read in ASCII format.

```
S1INDATA  DB   S1F.READ + S1F.FASC + S1F.RND
          DB   0
          DB   LU
          DB   0
          DA   INBUFF
          DA   256
          DA   0
          DA   1
          DA   0
SVC 1,S1INDATA
```

Ex. 2

Implement a Write.

```
NEWF      DB    S1F.WRITE
          DB    0
          DB    2
          DB    0
          DA    NEWFTXT
          DA    NEWFTXTL
NEWFTXT   DB    'NEWFILE.',7
NEWFTXTL  EQU   *-NEWFTXT
SVC 1, NEWF
```

8.4 ACCESS MODES

All input/output requests are made through SVC 1. All devices and files support binary transfer, proceed I/O, wait for completion, sequential access, and conditional proceed unless specified otherwise in the individual driver of file handler descriptions. The supported attributes that are listed in each description should be added to the previous list. The device code, which is a number between 0 and 255, defines all supported devices. Files, both contiguous and indexed, are available on each direct-access device supported by a given driver.

Indexed and Contiguous Files are handled in exactly the same way when performing both input and output. The main difference between them is that a contiguous file cannot be expanded while an indexed file can. Every file can be accessed in any of three different ways; namely as a:

1) Physical File - In 256 byte sectors.

2) Logical File - With fixed or variable record lengths.

3) Byte - As a stream of bytes.

The way in which the file is described is specified in SVC 7 when you open it. Note SVC 7 allows for an Access Mode (see S7.TAM in SVC 7) in the high nibble. This high byte allows for either physical access, logical access, or byte access.

1) Physical Access

When data is transferred as physical records no buffering or formatting is needed because the data is transferred in 256 byte blocks. When data is transferred as logical records an additional buffer (1 to 65,535 bytes) is used. Note that in this case the File Manager can handle partial sectors that belong to the same logical record but to different physical disk sectors (see Logical Access).

Read/Write

0)	S0F.WAIT	Standard.
1)	S1F.READ	Data is read from the current sector for "N" sectors.
2)	S1F.WRIT	Data is written into the current sector for "N" sectors.
3)	S1F.WRD	Data is written as S1F.WRIT and then verified. The sector index prints to the next sector when the operation is complete.

Data Type

2)	S1F.IBIN	Image binary. All others not supported.

Access Type

0)		Sequential. Sector index updated after each access. (No parameter needed)
1)	S1F.RND	Random access. Sector index updated to random address before access and updated after each access.

Special Operations

0)	SOF.TST	Standard
1)	SOF.CAN	Standard
2)	S1F.FR	Forward Record
4)	S1F.WF	Write File Mark. Updates EOF marker. Space allocated past this marker at this time is returned to the system.
5)	S1F.BR	Backward record if not a sector 0.
7)	S1F.RW	Sector index set to 0
9)	S1F.FEOF	Fetch EOF marker (last sector) into random address

All others are undefined.

S1F.WRIT, WRITE - Variable Record Length: Data is written on the disk from the current file pointer.

2) Logical Access

This mode is used to provide device independent I/O. It uses sytem buffering which means that a system buffer is allocated for each file. The system bufer is the size of a device sector (typically 256 bytes). Although the device is accessed on physical sector boundaries the program can access data as logical records. Data is read/written from the users buffer to the system buffer and then to the device with the system performing all additional manipulation to map logical records into physical sectors.

The logical records can be either of fixed length or variable length. A fixed length record is mapped contiguously into the physical sectors one record after another. For example, two 384 byte logical records map into three 256 byte physical sectors. Varioable record length files have the record length specified whenever a read or write operation is performed. The record size being the buffer size. Positioning is done on byte boundaries except for the special operations where the record size is 256 bytes.

Logical access can be used with all read/write operations and with:

0)	S1F.IASC	Image ASCII
1)	S1F.FASC	Formatted ASCII
2)	S1F.IBIN	Image Binary

ASCII transfers use space compression (bit 7 = 1 and bits 0 to 6 equal to the number of blanks). Records are terminated by 0.

When performing input operations where the buffer size is greater than the record size there are no fill characters and larger records are truncated. Output operations with a buffer size greater than the record length are truncated while a smaller buffer size is padded with zeroes for Binary and spaces for ASCII.

Operations are the same as for physical access with the appropriate modifications mentioned above.

3) Byte Access

Byte access is the same as logical access with variable length records.

SECTION 9
SVC 2 SUBFUNCTIONS

SECTION 9
SVC 2 SUBFUNCTIONS

9.1 INTRODUCTION

There are a number of service functions called subfunctions which are provided to the user. These are implemented by SVC 2.1 through SVC 2.12. These subfunctions are related to the tasks communication with the console operator, to memory allocation, to text processing, and to command processing.

The purpose of the function code for the subfunctions is generally to modify the conditions of the SVC call. The content of the function code is ignored by those subfunctions for which no function code has been defined. This means that no wait for completion and no unconditional proceed are supported. All subfunction requests require a Parameter Block accompanying the request.

The Parameter Blocks for the various SVC 2 subfunctions have the general form:

(0) SO.FC Function code	(1) SO.RS Return status
(2) S2.SNR Subfunction	(3) S2.PAR
(4) See SVC 2 subfunctions	

The parameter block for the SVC 2 subfunctions has the following structure:

	Offset	Bytes	Type	Mnemonic	Name
1)	0	1	Byte	SO.FC	Function Code
2)	1	1	Byte	SO.RS	Return Status
3)	2	1	Byte	S2.SNR	Subfunction
4)	3	1	Byte	S2.PAR	Parameter
	Total	4			

9.2 PARAMETERS

Descriptions of the parameters in the parameter block are given below:

1) S0.FC, Function Code

This parameter is determined by each subfunction.

2) S0.RS, Return Status

The Return Status is also determined by each subfunction. There is one common value, S2S.ISB, whose status code is 20 and which corresponds to an invalid subfunction number. S2S.ISB part of the Return Status Parameter of all SVC's. If a subfunction is used illegally, it is this status that will be returned.

3) S2.SNR, Subfunction

There are eight subfunctions available whose characteristics are as follows:

Subfunction	Status Code	Meaning
EV2.1MEM	1	Memory allocating.
EV2.2MSG	2	Log message.
EV2.3PFD	3	Pack file descriptor.
EV2.4PNU	4	Pack numeric data.
EV2.5UNP	5	Unpack binary number.
EV2.7DAT	7	Fetch or set date and time.
EV2.8CMD	8	Scan mnemonic table.
EV2.12OC	12	Open/close device.

4) S2.PAR, Other Data

The content of this field depends on each subfunction and is different for each one.

9.3 SVC 2.1 MEMORY HANDLING

SVC 2.1 is used to allocate and deallocate memory. The storage is allocated in system memory. Free space is allocated from the system table area (first 16K) and is very limited. Allocation is done on a first fit basis and the Operating System keeps a record of the size of the space that is actually allocated.

When releasing memory, only the location of the block to be released is required. Blocks of memory are released in the same fashion as they are accessed. For example, you cannot allocate a 2K byte block of memory and then release it in two 1K byte blocks. You must release it as a 2K byte block. Furthermore, deallocation must be done explicitly. The Operating System does support compaction.

Parameter Block

The Parameter Block for SVC2.1 is shown below.

(0) S0.FC Function code	(1) S0.RS Return status
(2) S2.SNR = 1	(3) S2.PAR Reserved
(4) S2.1ADR Memory address	
(6) S2.1SIZ Memory size	

The Parameter block for SVC2.1 has the following structure:

	Offset	Bytes	Type	Mnemonic	Name
1)	0	1	Byte	S0.FC	Function Code
2)	1	1	Byte	S0.RS	Return Status
3)	2	1	Byte	S2.SNR	Subfunction = 1
4)	3	1	Byte	S2.PAR	Reserved (not used) = 0
5)	4	2	Address	S2.1ADR	Memory Address
6)	6	2	Integer	S2.1SIZ	Memory Size
	Total	8			

Parameter

Descriptions of the parameter in the parameter block are given below.

1) S0.FC, Function Code

Memory Handling: The Memory Handling field is dependent upon the values of bits 0-2 as follows:

Bit	Value	Name	Description
0-2			Memory Handling Field
	1	S2F.1ALO	Allocate Memory
	2	S2F.1MAX	Reserved
	3	S2F.1REL	Release Memory
	4	S2F.1TCB	Allocated a Task Control Block (TCB). Only for internal use.
	5	S2F.1CAN	Remove a callers Task Control Block (TCB). Only for internal use.

2) S0.RS, Return Status: The Return Status Codes for SVC2.1 take on the values 21 and 22 as given below.

Function	Code	Meaning
S2S.1PAR	21	Illegal parameter.
S2S.1EOM	22	End of memory.

5) S2.1ADR, Memory Address: Contains the memory address at deallocation and returns the memory address at allocation.

6) S1.ISIZ, Memory Size: Specifies the memory size to be allocated in bytes.

9.4 SVC 2.2 LOG MESSAGE

SVC 2.2 is used to log a message on the terminal device or system log device irrespective of the logical unit assignments in force at the time of the request.

Parameter Block

The Parameter Block for SVC 2.2 is shown below.

(0) SO.FC	(1) SO.RS
Function code	Return status
(2) S2.SNR	(3) S2.2TS
Subfunction 2	Term. status
(4) S2.2BAD	
Buffer address	
(6) S2.2BSZ	
Buffer size	

The Parameter Block for SVC 2.2 has the following structure:

	Offset	Bytes	Type	Mnemonic	Name
1)	0	1	Byte	SO.FC	Function Code
2)	1	1	Byte	SO.RS	Return Status
3)	2	1	Byte	S2.SNR	Subfunction = 2
4)	3	1	Byte	S2.2TS	Reserved = 0
5)	4	2	Address	S2.2BAD	Buffer Address
6)	6	2	Integer	S2.2BSZ	Buffer Size
	Total	8			

Parameters

Descriptions of the parameters in the parameters block are given below.

1) SO.FC, Function Code: Refer to the SVC 1 Function Code parameters regarding data formatting.

2) SO.RS, Return Status: There is no return status.

3) Reserved.

4) S2.2BAD, Buffer Address: The address of the buffer to write on the system console.

6) S2.2BSZ, Buffer Size: This parameter specifies the number of bytes to write.

Example

```
        Log the message 'Records Copied' onto the console.
S22ANTR     DB      0
*           DB      0
            DB      EV2.2MSG
            DB      0
            DA      ANTRTXT
            DA      ANTRSIZ
*
ANTRTXT     DB      '   Records  Copied.
ANTRSIZ     EQU     *-ANTRTXT
SVC                 2,S22ANTR
```

9.5 SVC 2.3 PACK FILE DESCRIPTOR

SVC 2.3 allows the user to process a File Description in standard Monroe syntax. The scan proceeds until it has satisfactorily processed each field in the File Descriptors syntax. Headings and spaces are ignored. If the scan finds illegal characters, a syntax error is returned and the scan terminates. Note that some kind of termination character must exist if the string size is not specified.

Parameter Block

The Parameter Block for SVC 2.3 is shown below.

(0) S0.FC	(1) S0.RS	(2) S2.SNR=3	(3) S2.3TS
Function code	Return status		Term. status
(4) S2.3ADR		(6) S2.3BUF	
ASCII-string address		Address of receiving area	
(8) S2.3PNT		(10) S2.3CNT	
Terminating string address		String size	

The Parameter Block for SVC 2.3 has the following structure:

	Offset	Bytes	Type	Mnemonic	Name
1)	0	1	Byte	S0.FC	Function Code
2)	1	1	Byte	S0.RS	Return Status
3)	2	1	Byte	S2.SNR	Subfunction = 3
4)	3	1	Byte	S2.3TS	Termination Status
5)	4	2	Address	S2.3ADR	ASCII-String Address
6)	6	2	Address	S2.3BUF	Address of Receiving Area
7)	8	2	Address	S2.3PNT	Terminating String Address
8)	10	2	Integer	S2.3CNT	String Size
	Total	12			

Parameters

Descriptions of the parameter in the parameter block are given below.

1) S0.FC, Function Code: The function code byte allocation is given as follows:

Bit	Value	Name	Description
0	1	S2F.3FN	Unpack as filename if not specified.
1	2	S2F.3KEP	Keep non-modified fields.
2	4	S2F.3CNT	String size specified.
3	8	S2F.3PMO	Pack modifier.

2) S0.RS, Return Status: The Return Status parameters are as follows:

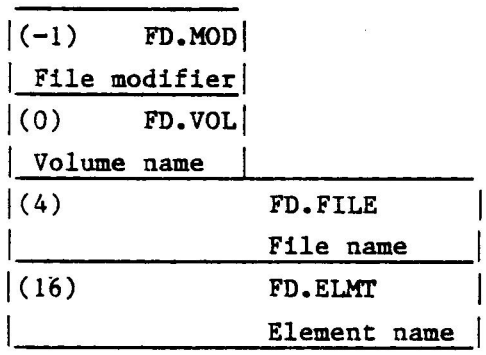
Function	Code	Meaning
S2S.3IFD	21	Invalid file descriptor, syntax error.

4) S2.3TS, Termination Status: The termination status byte allocation is given in the following table:

Bit	Value	Name	Description
0	1	S2T.3NEL	Element name not found.
1	2	S2T.3NFN	Filename not found.
2	4	S2T.3NVO	Volume name not found.
3	8	S2T.3NMO	Modifier not found. This status is only set if the pack modifier field S2F.3PMO is requested.

5) S2.3ADR, String Address: Is a pointer to a string that contains the file descriptor to be packed.

6) S2.3BUF, Receiving Area: This is a pointer to a 29-byte area. Note that the modifier field is on the negative side of the area, and must only be present if the function S2F.3PMO is requested. The following parameters are then included in the Parameter Block:



The modifier field has the following structure:

Offset	Bytes	Type	Mnemonic	Name
-1	1	Byte	FD.MOD	File Modifier
0	1	Byte	FD.VOL	Volume Name
4	2	Address	FD.FILE	File Name
16	2	Byte	FD.ELMT	Element Name

7) S2.3PNT, Terminating String Address: This field is returned pointing to the first byte that is not part of the file descriptor.

8) S2.3CNT, String Size: This is an optional field, specifying the string length. The lenght of this string can be limited by setting the S2F.3CNT-bit in S0.FC field, and give the size in S2.3CNT-field. A string length of zero means that the source string is terminated with 0.

9.6 SVC 2.4 PACK NUMERIC DATA

SVC 2.4 translates ASCII hexadecimal, decimal, or octal character strings into binary 8, 16, 24, or 32 bit numbers. Leading spaces are ignored and the conversion continues until a character not conforming to the base is found.

Parameter Block

The parameter block for SVC2.4 is shown below.

(0) SO.FC	(1) SO.RS	(2) S2.SNR = 4	(3) S2.4SIZE
Function code	Return status		Size
(4) S2.4ADR		(6) S2.4PNT	
String address		Updated string address	
(8) S2.4RES			
Result			

The parameter block for SVC 2.4 has the following structure:

	Offset	Bytes	Type	Mnemonic	Name
1)	0	1	Byte	SO.FC	Function Code
2)	1	1	Byte	SO.RS	Return Status
3)	2	1	Byte	S2.SNR	Subfunction = 4
4)	3	1	Integer	S2.4SIZE	Size
5)	4	2	Address	S2.4ADR	String Address
6)	6	2	Address	S2.4PNT	Updated String Address
7)	8	4	Byte	S2.4RES	Result
	Total	8			

Parameters

Descriptions of the parameters found in the parameter block are given below:

1) S0.FC, Function Code: The function code byte allocation is as follows:

Bit	Value	Name	Description
0-2	-	-	Conversion Base.
	0	S2F.4DEC	Decimal.
	1	S2F.4OCT	Octal.
	2	S2F.4HEX	Hexadecimal.
3			Sign Handling.
	0	S2F.4SGN	No sign input allowed.
	1	S2F.4SGN	Input may be signed.
4			Destination.
	0	2SF.4IND	In parameter block.
	1	2SF.4IND	Address Specified.

2) S0.RS, Return Status: The Return Status codes can take on the values of 21 and 22 as shown below.

Function	Code	Meaning
S2S.4OFL	21	Overflow.
S2S.4NCV	22	Nothing converted.

3) S2.4SIZE, Size: This parameter describes the size of the binary result. The user can auto-increment the result pointer using this parameter. The bit contents is:

Bit	Value	Name	Description
4-6	-	-	Size of result field.
		S2Z.4BIN	Contains the number of bytes in the result field.
7	-	-	Result poitner field.
	0	S2Z.$INC	Do not auto increment the result pointer.
	1	S2Z.4INC	Auto increment the result pointer.

Note that the size of the buffer through which the translation is invoked must be specified in advance. S2Z.4BIN then indicates the size of the binary result. If for example, you have characters in an array, that must be translated into binary S2Z.4BIN can be used to position the buffer into segments of some fixed memory length. Hence, if S2Z.4BIN is set at K bytes then a 1 in bit 7 for S2Z.4INC means that each time the result pointer is incremented it will do so in K byte segments of memory. During the actual data translation phase this removes an extra program step since you do not have the extra statement incrementing the result pointer after the previous array element has been buffered.

5) S2.4ADR, String Address: This is a poninter to the first character of the ASCII string to be converted.

6) S2.4RES, Result: The result is placed either in this field, or at the address specified by this field.

7) S2.4PNT, Updated String Address: This is the updated string pointer at return, and it is pointing at the first byte in the string that was not convereted.

Ex. 1

This example converts ASCII decimal to a binary number.

```
SVC24     DB      S2F.4DEC
STAT      DB      0
SUB       DB      EV2.4PNU .
SIZE      DB      STRLEN
STRAD     DA      STRDATA
STRRS     DA      0
RESBIN    DMA     2,0
STRDATA   DB      '1234567'
STRLEN    EQU     *-STRADATA
SVC               2,SVC2.4
```

9.7 SVC 2.5 UNPACK BINARY NUMBER

SVC 2.5 translates an 8, 16, 24 or 32 bit number into ASCII, hexadecimal, decimal, or octal format.

Parameter Block

The parameter block for SVC2.5 is shown below.

(0) S0.FC	(1) S0.RS	(2) S2.SNR = 5	(3) S2.5SIZE
Function code	Return status		Size
(4) S2.5ADR		(6) S2.5PNT	
Destination address		Updated string address	
(8) S2.5VAL			
Source			

The Paramter Block for SVC 2.5 has the following structure:

	Offset	Bytes	Type	Mnemonic	Name
1)	0	1	Byte	S0.FC	Function Code
2)	1	1	Byte	S0.RS	Return STatus
3)	2	1	Byte	S2.SNR	Subfunction = 5
4)	3	1	Integer	S2.5SIZE	Size
5)	4	2	Address	S2.5ADR	Destination Address
6)	6	2	Address	S2.5PNT	Updated String Address
7)	8	4	Byte	S2.5VAL	Source
	Total	12			

Parameters

Descriptions of the parameters are given below.

1) S0.FC, Function Code: The function code byte allocation is as follows:

Bit	Value	Name	Description
0-1			Base field.
	0	S2F.5DEC	Decimal.
	1	S2F.5OCT	Octal.
	2	S2F.5HEX	Hexadecimal
2			Sign field.
	0	S2F.5SGN	Unsigned conversion.
	1	S2F.5SGN	Signed conversion
3			Source description
	0	S2F.5IND	Number in parameter block.
	1	S2F.5IND	Address specified.
4			Space flag.
	0	S2F.5SP	Leading zeros.
	1	S2F.5SP	Leading spaces.
5			Field Justification.
	0	S2F.5LFT	Right Justify.
	1	S2F.5LFT	Left Justify.

Note that when converting a number to hexadecimal, decimal, or octal the base must be specified.

2) S0.RS, Return Status: There is no return status.

3) S2.5SIZE, Size: Describes the size of both the binary and ASCII fields, and allows you the possibility of auto-incrementing the binary pointer. If the number to be converted exceeds the buffer length, most of the significant bytes are lost. If suppression of leading zeros is requested, the number is stored in the buffer, and the remaining characters, if any, are filled with spaces. If the number is to be left justified, only the number of bytes required for the number will be used,

and S2.5PNT will point at the next position after the number. The byte allocation size is as follows:

Bit	Value	Name	Descripton
0-3			Bytes in ASCII.
	-	S2Z.5ASC	Contains the number of bytes in an ASCII character string.
4-6	-	S2Z.5BIN	Contains the number of bytes in a binary number.
7			Auto increment.
	0	S2Z.5INC	Do not auto increment binary pointer.
	1	S2Z.5INC	Auto increment binary pointer.

5) S2.5ADR, Destination Address: This is a pointer to the first location of a buffer in memory where the converted number is to be stored. This buffer must be in a writable logical segment.

6) S2.5PNT, Updated Destination Address: This is the updated buffer address pointer at return, and it is pointing at the first byte in the buffer after the converted number.

7) S2.5VAL, Source: The binary number is placed either in this field, or at the address specified by this field.

Ex.

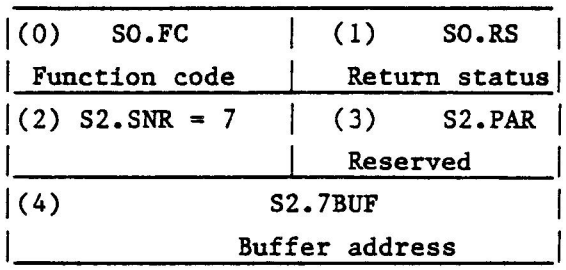
```
S25INERR  DB    S2F.5DEC + S2F.5SP
          DB    0
          DB    EV2.5UNP
          DB    10H+3
          DA    INERR
          DA    0
          DA    0
          DA    0
SVC 2,S25INERR
```

9.8 SVC 2.7 FETCH/SET OR DATE/TIME

SVC2.7 is used either to interrogate the time-slice value or to fetch and set the current time of day in the operating system.

Parameter Block

The parameter block for SVC2.7 is shown below.



The Parameter Block for SVC2.7 has the following structure:

	Offset	Bytes	Type	Mnemonic	Name
1)	0	1	Byte	SO.FC	Function Code
2)	1	1	Byte	SO.RS	Return Status
3)	2	1	Byte	S2.SNR	Subfunction = 7
4)	3	1	Byte	S2.PAR	Reserved
5)	4	2	Address or Integer	S2.7BUF	Byte for address or time slice value
	Total	6			

Parameters

The parameters found in the parameter block are described below:

1) S0.FC, Function Code: the byte allocation for the function code is as follows (S2.PAR is not used).

Bit	Value	Name	Description
0-1			Fetch 1 set field
	0	Illegal	-
	1	S2F.7GET	Fetch function
	2	S2F.7SET	Set function
	3	Illegal	-
2-3			Slice/date/time field
	0	S2F.7SLC	Slice handling
	1	S2F.7DAT	Date handling
	2	S2F.7TIM	Time handling
	3	(S2F.7DAT	Date and time handling.
		+ S2F.7TIM)	The buffer has the date followed by the time.
4-5			Data field
	0	S2F.7ASC	ASCII data; slice handling only.
	1	S2F.7BIN	Binary data, time and/or date handling only.

Note in all of the above fields there is a 0 in bit 5.

2) S0.RS, Return Status: The following return status codes are in effect.

Function	Code	Meaning
S2S.7DAT	21	Invalid date.
S2S.7TIM	22	Invalid time.

5) S2.7BUF, Buffer Address: This field 1) holds the value at slice handling. 2) Contains the address to a buffer within the user's program that either receives or sends the values at date and/or time handling.

If ASCII format is selected, the buffer must be ten bytes long for date handling, eight bytes long for time handling, and nineteen bytes long for both date and time handling. On setting the date and time the buffer must be terminated by a binary zero. The format is:

Date: "YYYY-MM-DD" (10 Bytes)
Time: "HH.MM.SS" (8 Bytes)

Date and time: "YYYY-MM-DD HH.MM.SS", 0 (19 Bytes)

Additional information regarding the parameter block fields are summarized in Table 9-2.

Table 9-2. SVC 2.7 Parameter Block Field

		S2.SNR	S2.PAR	S2F.7BUF
S2F.7SET + S2F.7TIM + S2F.7DAT	set time & date	U		U
S2F.7GET + S2F.7TIM + S2F.7DAT	get time & date	U		U
S2F.7SET + S2F.7SLC	set slice	U		U
S2F.7GET + S2F.7SLC	get slice	U		S

U - Should be initiated by user before SVC instruction.
S - Returned by system after SVC instruction.

9.9 SVC 2.8 SCAN MNEMONIC TABLE

SVC 2.8 permits the user to decode command mnemonics.

Parameter Block

The parameter block for SVC 2.8 is shown below.

(0) SO.FC Function code	(1) SO.RS Return status
(2) S2.SNR = 8	(3) S2.8INX Index
(4) S2.8ADR String address	
(6) S2.8LIST Mnemonic table address	
(8) S2.8PNT Updated string address	

The parameter block for SVC 2.8 has the following structure:

	Offset	Bytes	Type	Mnemonic	Name
1)	0	1	Byte	SO.FC	Function Code
2)	1	1	Byte	SO.RS	Return Status
3)	2	1	Byte	S2.SNR	Subfunction = 8
4)	3	1	Integer	S2.8INX	Index
5)	4	2	Address	S2.8ADR	String Address
6)	6	2	Address	S2.8LIST	Mnemonic Table Address
7)	8	2	Address	S2.8PNT	Updated String Address
	Total	10			

Parameters

The parameters for SVC 2.8 are described below.

1) SO.FC, Function Code: There is no function code for SVC 2.8.

2) SO.RS, Return Status: The following return status code is in effect.

Function	Code	Meaning
S2S.8CMD	21	Undefined command mnemonic.

4) S2.8INX, Index: This field returns the number in the table of the matched mnemonic, starting with zero. Thus, if a match is found on the third item in the table, the index returned is 2.

5) S2.8ADR, String Address: This field shall contain the address to the source string, within the user's program space, to be scanned.

6) S2.8LIST, Mnemonic Table Address: This field shall contain the address of a mnemonic table within the user's program space. A mnemonic table is composed of a string of mnemonics, separated from each other by a byte containing a binary zero. The end of the table is signified by the occurrence of two consecutive bytes of binary zeros.

The first byte of a mnemonic specifies the abbreviation size. The abbreviation must begin with the first character in the mnemonic, and must also be contiguous.

7) S2.8PNT, Updated String Address: This field returns the updated string address, and it is pointed to the first character that is not matched. This is normally a separator following the mnemonic in the string being scanned.

Illegal Characters

Characters not allowed in table words (terminal characters) are:

delete (7FH)
space (20H)
" # ' ( ) + , . / : ; < = > ?

Examples:

Ex. 1

Code the mnemonic "two words."

Note the letters TW and W are required. These are coded as:

```
CMDTABLE DB   2                        Abbreviation size.
         DB   'ONEWORD'                Command in ASCII.
         DB   0                        End of command.
         DB   2,'TWO',1,'WORDS',0
         DB   -1                       Flag to indicate that
         DA   NEXTABLE                 table continues at this
                                       address.
*
NEXTABLE DB   2,'TREE',1,'WORDS',1 'COMMAND',0
         DB   0                        End of table !
```

Ex. 2

This example shows how to decode command mnemonics.

```
SV2.8    DA 0,8
         DA STRADR
         DA TABADR

UPDTADR  DA 0
SVC      2,SV2.8
```

9.10 SVC 2.12 OPEN/CLOSE DEVICE

SVC 2.12 is used to take a device off-line or bring on-line a device that was previously off-line.

Parameter Block

The parameter block for SVC2.12 is shown below.

(0) SO.FC	(1) SO.RS	(2) S2.SNR = 12	(3) S2.PAR
Function code	Return status		Reserved
(4) S2.12FD		(6) S2.12ADR	
Name pointer, or device number		Optional SVC-handler address	

The parameter block for SVC2.12 has the following structure:

	Offset	Bytes	Type	Mnemonic	Name
1)	0	1	Byte	SO.FC	Function Code
2)	1	1	Byte	SO.RS	Return Status
3)	2	1	Byte	S2.SNR	Subfunction = 12
4)	3	1	Byte	S2.PAR	Reserved = 0
5)	4	2	Integer	S2.12FD	Name point, or device number
6)	6	2	Address	S2.12ADR	Optional SVC - handler address
	Total	8			

Parameters

The parameters for SVC 2.12 are described below:

1) S0.FC, Function Code: The function code byte allocation is as follows:

Bit	Name	Description
0	S2F.12CL	Close device
1	S2F.12OP	Open device
2	S2F.12PR	Write protected
3	S2F.NF	Nonfile structured
4	S2F.12AD	SVC handler address specified for directory oriented devices
5	S2F.12AL	Fetch auto start line. This is only valid when opening file structured files.

2) S0.RS, Return Status

The following return status codes are in effect:

Function	Code	Meaning
S2S.12AS	21	Device is assigned, can't be closed.
S2S.12DE	22	Device not found.
S2S.12IS	23	New volume already present.
S2S.12ON	24	Directory device not in close state.

5) S2.12FD, Name Pointer: This field either contains a pointer to a symbolic device name, or a numeric value less than 255 that is the numeric identity of the device.

6) S2.12AD, SVC-Handler Address: This optional field contains the address to a user written file handler.

S2F.120P, Function Open: If a directory device is opened file-structured by a symbolic name, the volume name will be returned in the file-name field of the file-descriptor.

S2F.12AL, Fetch Auto Start Line: Data contained in the auto start line area on the disk is returned to name pointer + 8. Eighty characters are moved.

Additional information regarding the parameter block fields are summarized in Table 9-3.

Table 9-3. SVC 2.12 Parameter Block Field

		S2.SNR	S2.PAR	S2.12FD	S2.12ADR
S2F.120P	OPEN	U		U/S	
S2F.12CL	CLOSE	U		U	
S2F.120P + S2F.12PR	OPEN WRITE PROTECTED	U		U/S	
S2F.120P + S2F.12NF	OPEN NON-FILE STRUCTURED	U		U	

U - Should be initiated by user before SVC instruction.

S - Returned by system after SVC instruction.

SECTION 10
SVC 3 TIMER REQUESTS

SECTION 10

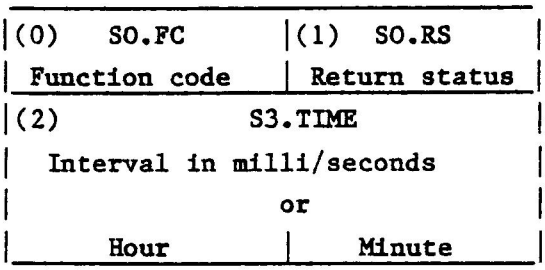
SVC 3 TIMER REQUESTS

10.1 INTRODUCTION

SVC 3 is used to coordinate tasks with the real time handler and supports both internal and time of day requests.

10.2 PARAMETER BLOCK

The parameter block for SVC 3 is shown below.



The parameter block for SVC 3 has the following structure:

	Offset	Bytes	Type	Mnemonic	Name
1)	0	1	Byte	SO.FC	Function Code
2)	1	1	Byte	SO.RS	Return Status
3)	2	2	Byte	S3.TIME	Time Interval
	Total	4			

Parameters

The parameters for SVC 3 are described below.

1) SO.FC, Function Code: The byte allocation for the function code is as follows:

Bit	Value	Name	Description
0-1			Time Specification
	1	S3F.MIL	Milliseconds
	2	S3F.SEC	Seconds
	3	S3F.TOD	Time of Day
0-2			Command Field
	0	SOF.TST	Test Request
	1	SOF.CAN	Reserved
	2	S3F.CMIL	Reserved
	3	S3F.CSEC	Reserved
	4	S3F.CTOD	Reserved
6			Wait Proceed bit

Note that in setting the time specification bits 2-5 are padded with zeros. In setting the command field bits 3-5 are set at 001.

2) S0.RS, Return Status: The following return status code is in effect.

Function	Code	Meaning
S3S.PAR	30	Invalid interval/time-of-day.

3) S2.3TIME, Interval: This field contains either an unsigned interval value, or an absolute time of day value.

SECTION 11
SVC 4 TASK DEVICE

SECTION 11
SVC 4 TASK DEVICE

11.1 INTRODUCTION

It is sometimes necessary for a symbiont task to retrigger one of its task devices in which case an item will be added to the event queue of the task. There are also times when it is necessary to ask the symbiont handler to cancel the request in progress. SVC 4 performs both of these functions and it is only used by the task that owns the task device.

11.2 PARAMETER BLOCK

The parameter block for SVC 4 is shown below.

(0) SO.FC	(1) SO.RS
Function code	Return status
(2) S4.LU	(3)
Logical unit	Reserved

The parameter block for SVC 4 has the following structure:

	Offset	Bytes	Type	Mnemonic	Name
1)	0	1	Byte	SO.FC	Function Code
2)	1	1	Byte	SO.RS	Return Status
3)	2	1	Integer	S4.LU	Logical Unit
4)	3	1	Byte	Reserved	
	Total	4			

11.3 PARAMETERS

The parameters for SVC 4 are described below.

1) S0.FC, Function Code

The byte allocation for the function code is as follows:

Bit	Value	Name	Description
0			Trigger field
	0	illegal	
	1	S4F.TRIG	Initial trigger
1			Cancel field
	0	illegal	
	1	S4F.CAN	Set cancel pending
0-1			Trigger + cancel field
	3	S4F.TRIG + S4F.CAN	Initiate trigger and set cancel pending

2) S0.RS, Return Status

The following return status codes are in effect:

Function	Code	Meaning
S4S.ASGN	40	Not assigned.
S4S.TYPE	41	Invalid device type.

3) S4.LU, Logical Unit

This field holds the logical unit that should be accessed.

SECTION 12
SVC 5 LOADER HANDLING

SECTION 12
SVC 5 LOADER HANDLING

12.1 INTRODUCTION

SVC 5 is normally used to load an overlay. It can alos be used to load a task although tasks are usually loaded through SVC6. An overlay is loaded into the requesting tasks segment at a relative address specified in the SVC 5 Parameter Block. An overlay can also be started at the same time as a task by changing the program execution to a new program segment. If the program which is loaded has an absolute code designation in the Loader Information Block (LIB) then it will be placed at the absolute location specified in the LIB and not at the buffer address in the SVC 5 Parameter Block.

However, if the program is relative relocatable code then it will be relocated and loaded into the buffer specified in the SVC 5 Parameter Block.

12.2 PARAMETER BLOCK

The parameter block for SVC 5 is shown below.

(0) S0.FC	(1) S0.RS	(2)	(3)
Function code	Return status	Reserved	Reserved
(4) S5.TID		(6) S5.LAD	
Name pointer or task number		Load address for overlay	
(8) S5.SAD		(10) S5.FD	
Start address		File descriptor	
(12) S5.SIZE			
Additional size			

The parameter block for SVC 5 has the following structure:

	Offset	Bytes	Type	Mnemonic	Name
1)	0	1	Byte	S0.FC	Function Code
2)	1	1	Byte	S0.RS	Return Status
3)	2	1		Reserved	
4)	3	1		Reserved	
5)	4	2	Integer	S5.TID	Name pointer or task number
6)	6	2	Address	S5.LAD	Load address for overlay
7)	8	2	Address	S5.SAD	Start Address
8)	10	2	Address	S5.FD	File Descriptor
9)	12	2	Integer	S5.SIZE	Additional Size
	Total	14			

12.3 PARAMETERS

The parameters for SVC 5 are described below.

1) S0.FC, Function Code

Bit	Value	Name	Description
0	1	S5F.LOAD	Load
1	1	S5F.STRT	Start Overlay
2	1	S5F.ABS	Absolute start
3	1	S5F.OVL	Overlay field Overlay handling or else task handling

Note that in each of the above fields bits 4 and 5 are padded with zeros. Each of the load, start, absolute start, and overlay fields is determined by a 1 in the indicated bit position. The remaining bits can take on either the value 0 or 1 depending upon the particular function or combination of functions desired.

2) S0.RS, Return Status

Function	Code	Meaning
S5S.TID	50	Illegal task name/number.
S5S.CODE	54	Illegal code/item at load.
S5S.SIZE	55	Overlay don't fit.

5) S5F.TID, Task Identifier

Values less than 256 are task numbers. Values over 255 print to a 4 byte area with the task name.

6) S5.LOAD, Load Overlay

The overlay is loaded from the device specified by the file descriptor, into the requesting tasks segment at a relativ address specified by S5.LAD. The calling program is placed in Load Wait until the overlay is loaded. If the overlay is successfully loaded, the root program may call it as a subroutine. Overlays can call other overlays directly; the calling code is overlaid with the new

overlay, but the task is aborted if the overlay load fails. Note that the overlay must fit within the root segments size.

7) S5.STRT, Start Overlay

This call starts the named overlay. If an absolute start is requested, the S5F.ABS-bit is set, and the overlay is started at the address specified by the S5.SAD field. If relative start is requested, the S6F.ABS-bit is cleared, and the overlay is started at its established transfer address plus the value in the S5.SAD-field. Additional information regarding the parameter block fields is summarized in Table 9-4.

Table 9-4. SVC 5 Parameter Block Fields

		S5.TID	S5.LAD	S5.SAD	S5.FD	S5.SIZE	
S5F.LOAD	LOAD	U	U		U		
	START						
S5F.STRT	OVERLAY		U	U	U		
	START +						
S5F.STRT + S5F.ABS	ABSOLUTE			U	U		

U - Should be initiated instruction.

8) S5.FD, File Descriptor

Prints to a 29 byte area with the file name (see SVC 7).

9) S2.SIZE, Additional Size

Additional area to be allocated to the task.

SECTION 13
SVC 6 TASK REQUEST

SECTION 13

SVC 6 TASK REQUEST

13.1 INTRODUCTION

SVC 6 is used to load tasks and control the interaction of a given task with other tasks. When using the event queue, any no wait SVC call will place the address of the no wait SVC parameter block on the event queue of the requesting task after the SVC is executed. A program therefore can suspend execution until all requests are completed.

The event queue for servicing of task devices can be implemented executing an SVC call. The SVC parameter block pointers are then added to the task's parameter event queue.

13.2 PARAMETER BLOCK

The parameter block for SVC 6 is shown below.

(0) SO.FC	(1) SO.RS	(2) S6.PRIO	(3) S6.OPT
Function code	Return status	Priority	Option
(4) S6.TID		(6) S6.PAR	
Name pointer or task number		Parameter	
(8) S6.SAD		(10) S6.FD	
Address		File descriptor	
(12) S6.SIZE			
Additional size			

The parameter block for SVC 6 has the following structure:

	Offset	Byte	Type	Mnemonic	Name
1)	0	1	Byte	SO.FC	Function Code
2)	1	1	Byte	SO.RS	Return Status
3)	2	1	Byte	SO.PRIO	Priority
4)	3	1	Byte	S6.OPT	Option
5)	4	2	Integer	S6.TID	Name Pointer or task number
6)	6	2	Byte	S6.PAR	Parameter
7)	8	2	Address	S6.SAD	Address
8)	10	2	Address	S6.FD	File descriptor
9)	12	2	Integer	S6.SIZE	Additional size
	Total	14			

13.3 PARAMETERS

The parameters for SVC 6 are described below:

1) S0.FC, Function Code

The upper two most significant bits are used for the wait function and the unconditional proceed function. The remaining biits are encoded into the reamining six bits. Any combination not listed below is illegal.

Bit 0-5

Value (Hex)	Name	Note	Description
01	S6F.LOAD	1	Load Task
02	S6F.STRT	1	Start Task
04	S6F.ABS	1	Absolute Start Address
08	S6F.QTST		Test Event Queue
09	S6F.QWAI	2	Wait on Event Queue
0A	S6F.QTRM	2	Terminate Event
0C	S6F.QDIS		Disable Event Queue
0D	S6F.QENI		Enable Event Queue
0E	S6F.SUSP		Suspend Myself
20	S6F.TST		Test Task
21	S6F.CAN		Cancel Task
22	S6F.PAUS		Pause Task
23	S6F.CONT		Continue Task
25	S6F.PRIO	3	Set Task Priority
26	S6F.OPT	3	Set Task Options
28	S6F.TSKW		Wait for TASK termination
29	S6F.ADDQ		Add event to queue
2A	S6F.STSW		Wait for task status change
2B	S6F.TYPE		Set TASK type

NOTE 1: These can be or'ed together as one function.

NOTE 2: These can be or'ed together as one function.

NOTE 3: These can be or'ed together as one function.

A more complete descrition can be found in Section 13.4.

2) S0.RS, Return Status

Function	Code	Meaning
S6S.TID	60	Illegal task name/number.
S6S.PRES	61	Task already present.
S6S.PRI	62	Illegal priority.
S6S.OPT	63	Illegal option.
S6S.EQUE	64	Event queue disabled.
S6S.STAT	65	Invalid task status.
S6S.QPAR	66	Invalid termination parameter.
S6S.QITM	67	More items present in event queue.
S6S.TYPE	68	Invalid task type.

3) S6.PRIO, Task Priority

This field is used with function S6F.STRT of the function code. The value zero means that the priority defined at the time of task-establishment should be used.

4) S6.OPT, Task Option

The S6.OPT-field is used at functions S6F.OPT and S6F.TYPE of the function code and contains the options of the task.

Options at S6F.OPT

Bit	Name	Description
0	S60.DASG	Default assign allowed.
1	S60.NSTK	No stack check.

S6F.OPT: Options field for S6F.TYPE Function

Bit	Name	Description
0	S6T.RES	Set the task to be resident in memory.
1	S6T.NAB	Set the task to be non-addable from other tasks.

5) S6.TID, Task Identifier
Values less than 256 indicate a task number while values greater than 255 are pointers to four byte task name areas.

6) S6.PAR, Parameter
Function dependent.

7) S6.Address
Function dependent.

8) S6.FD
Prints to a 29 byte area containing a packed site name. (See SVC 7.)

9) S6.SIZE
Additional number of bytes to be allocated when a task is loaded.

13.4 FUNCTION CODE DESCRIPTION

The following functions can be input for the function code in SVC 6. Each function requires specific fields which are indicated under the function.

2) S6F.LOAD, Function Load Task (01 Hex)

The required fields are: S6.PRIO, S6.OPT, S6.TID, S6.FD and S6.SIZE.

The specified task is loaded from the device specified by S6.FD. If a task is already present in the system with the given S6.TID or if S6.TID is invalid, then the call is rejected. A memory area, expanded with S6.SIZE is allocated and the task is given the name found by the S6.TID field. If there is not memory enough, the Load Task call is rejected.

S6F.STRT, Function Start Task (02 Hex)

The required fields are: S6.PRIO, S6.TID, S6.PAR and S6.SAD.

This call, which can either be an absolute start or a relative start, begins the indicated task.

A) Absolute Start (S6F.STRT OR'ED WITH S6F.ABS)

.S6.SAD contains the address at which the task is started.

.S6.PRIO, if not zero, is the priority of the task.

.S6.PAR, if not zero, can be interpreted as the address to a vector whose first two bytes are the number of bytes that should be transferred to the task. These are stored on the task's stack. For an odd number of bytes, a last byte of zero is added.

B) Relative Start (S6F.STRT without S6F.ABS)

.The task is started at its established transfer address plus the value in the S6.SAD field.

.S6.PRIO, if not zero, is the priority of the task.

.S6.PAR, if not zero, can be interpreted as the address to a vector whose first two bytes are the number of bytes that should be transferred to the task. These are stored on the task's stack. For an odd number of bytes a last byte of zero is added.

C) Register Usage

For both absolute and relative start operations. The following register usage is indicated:

Register	Use
A	Transferred from the starting task.
B,C,D,E	32 bit switch pattern (A=bit 0 of register B)
H	Transferred from the starting task.
L	Transferred from the starting task.
Y	Points to the first byte after the program.
X	Points to the lowest possible value, -1, for the system pointer (SP).
SP	Top-of-stack

On the top of the stack is a 16-bit unsigned integer that holds the number of parameters bytes present in the stack. If no parameters are transferred, the value is zero. The size of the additional memory added to a task is calculated by subtracting register Y from register X.

S6F.QTST, Function Test Event Queue (08 Hex)
The required fields are: S6F.PRIO, S6F.TID and S6.PAR.

This request is used to test if there is anything in the event queue. If empty, the request returns with a return status of zero, otherwise, it returns with a return status of S6S.QITM where the contents of the fields are valid. This call will not remove the event from the event queue.

S6F.QWAI, Function Wait for Event (09 Hex)
The required fields are: S6.PRIO, S6.TID and S6.PAR.

This request is used when no more actions can be taken by the task. The task will enter the trap wait state. As soon as any item is added to the event queue, or if the event queue isn't empty, the task returns with a return status of zero, where all fields are valid. If the item belongs to an external task, S6.TID is non-zero and the item is saved in a slough queue within the task. The item will remain in the slough queue until it is terminated by the task. If the item is self-directed, S6.TID is zero, and it is not saved in the slough queue. The field S6.PAR holds the address of the SVC-parameter block that initiated the event. This field contains the node-address at an external event, and the item is not saved in the slough queue.

If the item added to the event queue refers to a task-device, then the symbiont initiator for that task-device will be entered, and the call will not return to the task. The symbiont handler works in the same way as a real device driver with the same conditions. The user's urged to consult the Section on Device Drivers in this manual for a more detailed account of the above.

S6F.QTRM, Function Terminate Event (0A Hex)
The required fields are: S6.PRIO and S6.PAR.

This function is used to remove an external request from the slough queue and to terminate it. S6.PRIO contains the final return status for that request, and S6.PAR contains the parameter received at S6F.QWAI. It is also possible to combine S6F.QTRM and S6F.QWAI.

S6F.QDIS, Function Disable Event Queue (0C Hex)
This function is used to close the event queue, and every present or queued request will be terminated with a return status of S0F.OFFL.

S6F.QENI, Function Enable Event Queue (0D Hex)
The event queue of a task must be opened prior to using the event queue. Any addition to a non-enabled event queue will fail, and those requests will have a return status of S0S.OFFL.

S6F.SUSP, Function Suspend Self (0E Hex)
The required field is: S6.PRIO.

This request is used to relinquish control of the processor and is added to the Ready-Queue. The priority to be used is specified in the field S6.PRIO. If the value is zero, the current priority will be used.

S6F.TST, Function Test Task (20 Hex)
The required fields are: S6F.PRIO, S6.OPT, S6.TID and S6.PAR.

This function is useful when you want to test for the presence of a task. In this case the TCB-address is returned in S6.PAR.

S6F.CAN, Function Cancel Task (21 Hex)

This call permits a task to terminate itself, or another task, in an orderly fashion. The Return Code of the task S6.PRIO, may be treated as required by some other task waiting for the termination of the initial task. Normally the return code 0 represents normal termination. If the task has I/O in progress at the time the call is made, the I/O must be completed before the task goes to the end of job. At this point all of its files and devices are closed. If the task is non-resident in memory, it is cancelled by deleting all control information pertaining to the task. The task will never return from the call if it is self-directed. The required fields are: S6.PRIO and S6.TID.

S6F.PAUS, Function Pause Task (22 Hex)

This function causes the specified task to enter the Pause state. The task does not continue to execute until it is released by an S6F.CONT call. A task may also suspend itself. In that case, another task must be available to subsequently release it. The required field is: S6.TID.

S6F.CONT, Function Continue Task (23 Hex)

This function continues a task, by taking a paused task out of its Pause state. The task will continue to execute as it did before it was paused, provided it is not in any other Wait state. The required field is: S6.TID.

S6F.PRIO, Function Change Priority (25 Hex)

The required fields are: S6.PRIO and S6.TID.

This function changes the priority of the specified task to that specified in the S6.PRIO field of the parameter block. The call is rejected if the priority is outside the valid range of 1-255.

S6F.OPT, Function Change Options (26 Hex)

This call will change the options on a task according to the bit pattern in S6.OPT. It is possible to combine S6F.PRIO with S6F.OPT. The required fields are: S6.OPT and S6.TID.

S6F.TSKW, Function Wait for Task Termination (28 Hex)

This call is used when a task wants to wait for another task and its termination. At return from the call, S6.PRIO holds the new status of task, S6.PRIO contains the return code from the task. The required fields are: S6.PRIO, S6.OPT and S6.TID.

S6F.ADDQ, Function Add Event to Queue (29 Hex)

The parameter in the S6.PAR field of the parameter block is added to the specified task's Event Queue if the queue is enabled. Otherwise, the call is rejected with appropriate error status. The required fields are: S6.TID and S6.PAR.

S6F.TSTW, Function Wait for Task Status Change (2A Hex)

This call is used by a task, when the task wants information about any status change on another task. At return from the call, the fields contents are the same as on S6F.TSKW. The status changes are:

- .Transition to/from Dormant state.
- .Transition to/from Pause state.

The required fields are: S6.PRIO, S6.OPT and S6.TID.

S6F.TYPE, Function Change Task Type (2B Hex)

This call will change the type of a task according to the bit pattern in S6.OPT. The required fields are: S6.OPT and S6.TID.

13.5 EVENT QUEUE HANDLING

The following functions describe how to handle the event queue. In general, every no-wait request will be added to the event queue if it is enabled. The required fields are almost always: S6.PRIO, S6.TID and S6.PAR. The contents of the fields, if valid are:

Valid	Content
S6.PRIO	Contains either the termination status on S6F.QTRM or else the SVC type.
S6.TID	Contains the task number which is zero if the task is self directed.
S6.PAR	Contains the address of the invoked parameter block.

All of the information regarding the Parameter Block fields are summarized in Table 13-1.

Table 13-1. - SVC 6 Parameter Block Field

	S6.PRIO	S6.OPT	S6.TID	S6.PAR	S6.SAD	S6.FD	S6.SIZE
S6F.LOAD	U	U	U			U	U
S6F.STRT	U	U	U	U	U		
S6F.QTST	S=SVCNR		S=TNR	S=PBLK			
S6F.QWAI	S=SVCNR		S=TNR	S=PBLK			
S6F.QTRM	U=SO.RS			U=PBLK			
S6F.SUSP	U						
SOF.TST	S	S	U	S=TCB	S		
SOF.CAN	U=RCOD		U				
S6F.PAUS			U				
S6F.CONT			U				
S6F.PRIO	U		U				
S6F.OPT		U	U				
S6F.TSKW	S=STAT	S=RCOD	U				
S6F.ADDQ			U	U=PBLK			
S6F.STSW	S=STAT	S=RCOD	U				
S6F.TYPE		U	U				

U - Should be initiated by user before SVC instruction.
S - Returned by system after SVC instruction.
PBLK - Address of invoked parameter block.
RCOD - Return code at task termination.
SO.RS - Final return status for the request.
STAT - New task status.
SVCNR - Type of SVC.
TCB - Address of Task Control Block.
TNR - Task number, zero if self directed.

SECTION 14
SVC 7 FILE REQUEST

SECTION 14

SVC 7 FILE REQUEST

14.1 INTRODUCTION

SVC 7 is used to create and assign files, devices, and tasks to a logical unit. It can also be used to modify already existing file, device, and task assignments. The meaning and use of each field in the Parameter Block is described by the function requiring that field. When dealing with nonrandom access devices you do not need to specify the record length or size fields.

14.2 PARAMETER BLOCK

The parameter block for SVC 7 is show below.

(0) SO.FC	(1) SO.RS	(2) S7.LU	(3) S7.MOD
Function code	Return status	Logical Unit	Modifier
(4) S7.FD		(6) S7.CLAS	(7) S7.TAM
Name pointer or device number		Class	Access mode
(8) S7.RECL		(10) S7.UBUF	
Record length		Reserved	
(12) S7.SIZE			
(LSW) Size		(MSW)	

The parameter block for SVC 7 has the following structure:

	Offset	Bytes	Type	Mnemonic	Name
1)	0	1	Byte	SO.FC	Function Code
2)	1	1	Byte	SO.RS	Return Status
3)	2	1	Intéger	S7.LU	Logical Unit
4)	3	1	Byte	S7.MOD	Modifier
5)	4	2	Address	S7.FD	New pointer on device number
6)	6	1	Integer	S7.CLAS	Class
7)	7	1	Byte	S7.TAM	Access mode
8)	8	2	Integer	S7.RECL	Record Length
9)	10	2	Address	S7.UBUF	Reserved
10)	12	4	Long Int.	S7.SIZE	Size
	Total	16			

14.3 PARAMETERS

The parameters for SVC 7 are described below:

1) SO.FC, Function Code

Bit	Value Hex	Name	Description
			Open/Close Field (Bits 5 = 0)
0		S7F.ALLO	Allocate
1		Reserved	Reserved
2		S7F.ASGN	Assign (Open)
3		S7F.DELL	Delete at close
4		S7F.CLOS	Close
			Test/check field
0-5	20	SOF.TSTS	Test request
	21	SOF.CAN	Cancel all previous requests
	22	Reserved	Reserved
	23	S7F.CHKP	Check point
	24	Reserved	Reserved
	25	Reserved	Reserved
	26	S7F.RNAM	Rename
	27	S7F.FAT	Fetch attributes

Note that the test/check field is dependent upon the values of the bits 0-2. Bits 3-5 are padded with 001. In the Allocate Reserved, Assign, Delete, and Close fields bit 5 is set to zero.

2) S0.RS, Return Status

The following return status codes are in effect:

FUNCTION	CODE	MEANING
S7S.ASGN	70	Assignment error.
S7S.AM	71	Illegal access mode.
S7S.SIZE	72	Size error, size field invalid or specifies non existing space.
S7S.TYPE	73	Type error, LU is not a direct-access device.
S7S.FD	74	File descriptor error, file descriptor of invalid format.
S7S.NAME	75	Name error, matching name not found.
S7S.KEY	76	Invalid key.
S7S.FEX	77	File already exist.
	1x	I/O error, SVC1 code given.

3) S7.LU, Logical Unit

The Logical unit number is assigned by the user.

4) S7.MOD, Modifier

The modifier specifies the type of a file at allocation and assignment time. The data formats are described in the section titled FILEFORMAT. The file type is divided into two 4-bit groups (nibbles):

5) S7.FD, Name Pointer

This address byte points to the file name. (See file descriptor formats.) Values lower than 256 mean that the reference is made to a system device with the same number.

The most significant nibble specifies the type of data in the file:

Modifier (Bits 4-7)	Code	File Type
FCM.ASC	1	ASCII data readable without any special handling.
FCM.LIST	2	List file, ASCII data together with positioning information.
FCM.OBJ	3	Object code, readable by the ESTAB.
FCM.BIN	4	Binary data, which is unspecified.
FCM.TSK	5	Task file, either relocatable or absolute.
FCM.ISM	6	ISAM index file.
FCM.DIR	F	Directories.

The least significant nibble, if specified, defines a set of languages and directory types:

Modifier (Bits 0-3)	Code	FIle Type
FCM.ASM	1	ASEMBLER source code.
FCM.BAS	2	Monroe BASIC source code, or data produced by Monroe BASIC
FCM.COB	3	Reserved.
FCM.FTN	4	Reserved.
FCM.PAS	5	Monroe PASCAL source code, or data produced by Monroe PASCAL.

Some special modifiers are specified below:

Modifier	Code	File Type
FCM.UNDF	00	Undefined data, verifies to any other type.
FCM.EFD	FD	User File Directory.
FCM.MFD	FF	Master File Directory.

6) S7.CLAS, Class

This field specifies the resource class to be accessed.

Resource	Code	Meaning
S7C.ALL	000	Scan all resources.

7) S7.RECL, Record Length

This address field on the Allocate File, contains the logical record length for an Indexed File. If the field is zero, variable record length is assumed.

8) S7.SIZE, Size

This field is defined for the Allocate Call. It depends upon the type of file being allocated. When the high order bit in the field is set, the remaining bits express the size of a single data block or continuous file. When an indexed file is allocated, the Size field is divided into two 16-bit fields; the first field (LSW) contains the segment size in sectors, the second field (MSW) contains the number of segments to be preallocated. If the size field is set to zero in the case of an indexed file, default sizes are taken from the volume information. (See disk initialization.) Size is not used for Non-Direct-Access devices.

14.4 FUNCTION CODE DESCRIPTIONS

The following functions specified below can be used for the function code in SVC 7. Each function requires particular fields as described in Section 14.3.

S7F.ALLO, Allocate

This function allocates space for files.

SF7.ASGN, Assign

This function assigns a logical unit to a device or file.

S7F.DELC, Function Delete at Close

This function allows the user to delete a file on a direct-access device. When the call is completed the file is not deleted, only a flag in the FCB is set. The actual delete is performed when the file is closed. This feature allows for very simple Temporary File handling. The only parameter required is an LU.

Applicable error codes are:

Function	Code	Meaning
S7S.ASGN	70	LU Error.
S7S.AM	71	Protect Error, file not ERW assigned.
S7S.TYPE	73	Type Error, LU is non direct access device.
	1x	I/O Error, as returned by SVC1.

S7F.CLOS, Function Code

This function discontinues an assigned logical connection between a task and a file or a task and a device. The logical unit LU is the only required parameter. When entered the specified LU is then de-assigned. Logical units which have been assigned for Write access to files or buffered terminals will have any partially filled buffers written to the file by a CLOSE call. Direct access devices with the Delete-At-Close flag set will be deleted.

Applicable error codes are:

Error	Code	Meaning
S7S.ASGN	70	LU Error.
	1x	I/O Error, as returned by SVC 1.

S7F.CHKP, Function Checkpoint

The checkpoint function flushes the Operating System Management buffers and updates the File Information Block. LU is the only required parameter.

The user may wish to employ Checkpointing after sensitive data is added to a buffered file. Since logical blocking of data in the system buffers leaves the file vulnerable, the integrity of the data can be preserved on the direct-access device by Checkpointing. In case of a system failure, all data on Indexed 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. File name, access privileges, and keys need not be specified.

Applicable error codes:

Function	Code	Meaning
S7S.ASGN	70	LU Error.
	1x	I/O Error, as returned by SVC 1.

S7F.RNAM, Function Rename

This function changes the name of an assigned file. The file must currently be assigned to ERW. The required parameters are LU and the File-descriptor. the LU must be assigned to a direct-access file (unless the caller is an Executive Task which may rename a non direct-access device). The Volume field of the file descriptor is ignored. The specified File descriptor replaces the previous file descriptor in the directory if the rename function is successful. If the modifier field is 0, then either the modifier will not be changed, or it will be assigned the value in the parameter block.

Applicable error codes are:

Function	Code	Meaning
S7S.ASGN	70	LU Error.
S7S.AM	71	Access Mode Error.
S7S.TYPE	73	Type Error, LU is non direct-access.
S7S.FD	74	File Descriptor Error, file descriptor of invalid format.
S7S.NAME	75	Name Error, new name already exists.
	1x	I/O Error, as returned by SVC 1.

S7F.FAT, Function Fetch Attributes

Certain programs may require, for proper operation, knowledge of the physical attributes of the device or file associated with a given LU. The only required parameter is the Logical Unit. The system returns information in the field Modifier, Name pointer, Record length, User supplied buffer address, and Size.

The Modifier byte is set to indicate the file or device type.

The Name pointer must be an address to a buffer where the system returns the name or the mnemonic of the assigned resource. If the pointer value is less than 256, the device number is returned in this field, rather than the name.

The Record length field is set equal to the physical record length associatd with the resource.

For this call the reserved field S7.UBUF is redefined to receive the indicated attributes.

The current size of a direct-access file is returned in the Size field.

Applicable error codes are:

Function	Code	Meaning
S7S.ASGN	70	LU error, illegal LU or LU not assigned.

S7.TAM, Access Mode

The low nibble of this byte specifies the Access Priviliges. If the access mode for a direct access file is 'SW', it will be changed to 'EW' Note S7.TAM is used in conjunction with SVC 1 to define the access (hysical, logical, or byte) of a file. The available types of file access and their associated attributes are listed below:

Bit	Value	Name	Description
0-2			Access mode field
	0	S7A.SRO	Sharable read only
	1	S7A.ERO	Exclusive read only
	2	S7A.SWO	Sharable write only
	3	S7A.EWO	Exclusive write only
	4	S7A.SRW	Will position to EOF
	5	S7A.SREW	Sharable read write
	6	S7A.ERSW	Sharable read, exclusive write
	7	S7A.ERW	Exclusive read-write
6-7	-	-	Command modifier fields
	0	S7A.SBUF	System buffering required, device dependent I/O
	1	S7A.UBUF	Reserved
	2	S7A.PHYA	Physical access, access on disk sector level
	3	S7A.BYTE	Byte addressing I/O, file treated as a file of bytes

14.5 FILE FORMATTING

Much of the data formatting within a file is standardized. The formatting is described below.

ASCII Files, Compressed Variable Length Records:

- .Spaces are compressed to 80H + the number of spaces.
- .Records are separated by a NULL-byte.
- .Records are stored after each other without any padding, to minimize the storage required.

ASCII Files, Fixed Record Length:

- .Not separated by a null byte.

Binary Files, Fixed Record Length:

- .Records are stored continuously after each other.
- .The data bytes are not specified.
- .Both random and sequential access can be done.

Load Modules:

- .The data formatting is defined by the code-type.

Additional information regarding the parameter block fields is summarized in Table 14-1.

Table 14-1. SVC 7 Parameter Block Fields

	S7.LU	S7.MOD	S7.FD	S7.CLAS	S7.TAM	S7.RECL	S7.UBUF	S7.SIZE
S7F.ALLO	U	U	U	U=0 gen	U	optional	gen 0	gen 0
S7F.ASGN	U	U	U	U=0 gen	U			
S7F.CLOS	U			U=0 gen				
S7F.CLOS S7F.DELC	U			U=0 gen				
S7F.CHK		U						
S7F.RNAM	U	If 0,chg modifier	U		S7A.ERW			
S7F.FAT	U	S	S			S	S	S

U - Should be initiated by user before SVC instruction.

S - Returned by system after SVC instruction.

SECTION 15
SVC 8 RESOURCE HANDLING

SECTION 15

SVC 8 RESOURCE HANDLING

15.1 INTRODUCTION

SVC 8 is used to both establish and remove resources within the Operating System. It is normally used by system programers, because it requires a very good knowledge of the structure and functions of the Operating System.

The SVC 8 function creates a new resource from the free space located in the system table area (first 16KB) and added to the appropriate system list as designated by the class. The resource is initialized by the buffer area pointed at by S8.ADR. The class indicates the format of this buffer area. Note that items such as tasks have many fields described in the buffer area (RDT, TDT, and EDT for a task). These are concatenated in the buffer area to form a contiguous space. Note that the buffer does not always physically map one to one into the resource.

15.2 PARAMETER BLOCK

The parameter block for SVC 8 is shown below.

(0) S0.FC	(1) S0.RS	(2) S8.RNR	(3) S8.PRIO
Function code	Return status	Resource no.	Resource prio
(4) S8.ID		(6) S8.CLAS	(7) S8.TYPE
Name pointer or resource no.		Class	Type
(8) S8.ADR		(10) S8.CA	(11) S8.IL
Entry or 'RDT'-list address		S8.SIZE	

The parameter block for SVC 8 has the following structure:

	Offset	Bytes	Type	Name	Description
1)	0	1	Byte	S0.FC	Function Code
2)	1	1	Byte	S0.RS	Return Status
3)	2	1	Byte	S8.RNR	Resource Number
4)	3	1	Byte	S8.PRIO	Resource Priority
5)	4	2	Address	S8.ID	Name pointer or resource number
6)	6	1	Byte	S8.CLAS	Class
7)	7	1	Byte	S8.TYPE	Type
8)	8	2	Address	S8.ADR	Entry or 'RDT' list address
9)	10	1	Integer	S8.CA	Size
10)	11	1	Integer	S8.IL	Size

15.3 PARAMETERS

The parameters for SVC 8 are described below:

1) S8.FC, Function Code

Bit	Value	Name	Description
0-1	-	-	Resource Field
	0	illegal	
	1	S8F.EST	Establish resource
	2	S8F.RMOV	Remove resource, only available to the owner
	3	S8F.TEST	Test the presence of a resource
2	-	-	RCB status field
		S8.NRCB	RCB already present

2) S0.RS, Return Status

The following return status codes are in effect:

FUNCTION	CODE	MEANING
S8S.ID	80	Illegal name/number.
S8S.CLAS	81	Illegal class.
S8S.PRES	82	Already present.
S8S.PRNT	83	Parent not present.
S8S.DUAL	84	Dual not present.
S8S.RCB	85	Invalid 'RCB'-type.
S8S.EOM	86	End of memory.

3) S8.RNR, Resource Number

This field is used to specify the numeric identity of a resource. If the number zero is chosen, the system will select the first free number and the numeric identity will be returned in this field.

4) S8.PRIO, Priority of Number

This field contains either the task/device priority, or the parent/dual resource number.

5) S8.ID, Name Pointer

This field contains either a pointer to a 4 character symbolic name, or a numeric value less than 255, that is the identity of the resource.

6) S8.CLAS, Class

This field contains the resource class.

The byte allocation for the class field is as follows:

Bit	Value	Name	Description	S8.ADR Buffer Contents
0-2	-	-	Class field	
	0	illegal		
	1	S8C.DEV	Devices	(RDT,DDT, IDT.CDT)
	2	S8C.TSK	Tasks	(RDT,TDT)
	3	S8C.COM	Common	(RDT)
	4	S8C.VOL	Volumes	(RDT)
	5	S8C.SVC	SVC-functions	(RDT)
	6	S8C.SVC2	SVC-subfunctions	(RDT)

Note: All classes are terminated with an EDT.

7) S8.TYPE, Type

The field contains the resource type. The byte allocation for the resource type is as follows:

Bit	Value	Name	Description
0-1	-	-	Resource type field
	0	RTT.PURE	Shared resource field
	1	RTT.RCB	Exclusive resource
	2	RTT.RRT	Dummy resource S8.ADR points to a new resource
	3	RTT.AREA	Area, no entry
2		RTT.DIR	Directory oriented
3		RTT.SVC	Entries at all SVC-calls
4		RTT.OFFL	Offline, not accessible
5		RTT.PROT	Protected,write not allowed
6		RTT.NFST	Non file structured, only for systems use
7		RTT.NRMV	Resident, non removable

8) S8.ADR, Entry/RDT

This is either the entry address of a shared resource, or the address to a Resource Descriptor Table (RDT) for an exclusive resource.

9) S8.SIZE, Size

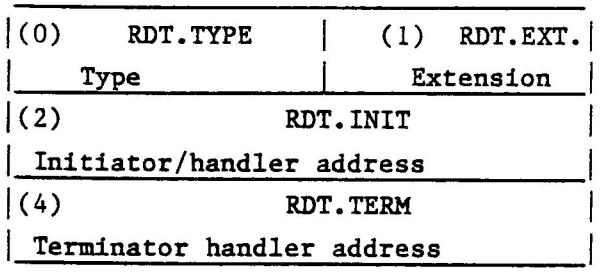
Contains the additional memory size which should be allocated at the time of task establishment.

S8.CA, Channel Select code: Contains the card identity for the interface.

S8.IL, Interrupt Level: Contains the interrupt level for a physical device.

15.4 RESOURCE DESCRIPTOR TABLE (RDT)

A RDT describes an exclusive resource, such as a Device Control Block (DCB), or a Task Control Block (TCB). This table is used by the SVC 8-handler to create the control blocks necessary to handle the resource. Each exclusive resource has at least an RDT. The parameter block for the RDT is shown below.



The parameter block for the RDT has the following structure:

	Offset	Byte	Type	Name	Description
1)	0	1	Byte	RDT.TYPE	Type
2)	1	1	Byte	RDT.EXT	Extension
3)	2	2	Address	RDT.INIT	Initiator/handler address
4)	4	2	Address	RDT.TERM	Terminator handler address

Parameters

The parameters for the RDT are described below:

1) RDT.TYPE, Type: The byte allocation for the type is as follows:

Bit	Value	Name	Description
0-2			Table field
	0	RCT.RCB	No special type
	1	RCT.DCB	DCB, device descriptor table present
	2	RCT.TCB	TCB, task descriptor table present
	3	RCT.FCB	Only for systems use
	4	RCT.VCB	VCB, volume descriptor table present
	5	RCT.ART	Area, no entry
3	1	RCT.PRNT	Coordination parent specified
4	1	RCT.DESC	Only for system's use
5	1	RCT.NW	Don't support "no-wait" function
6	1	RCT.PRO	Don't support unconditional proceed
7	1	RCT.NAB	Non-abortable, can't be cancelled

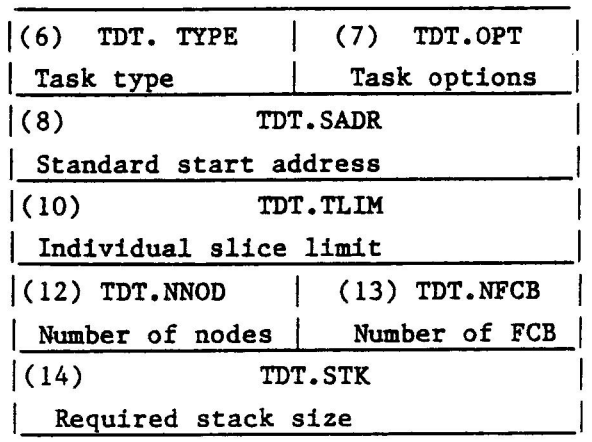
2) RDT.EXT, Extension: Makes it possible to expand any control block, i.e. expand a DCB, where the expansion is used by the device driver. Refer to the section entitled Extended Descriptor Table.

3) RDT.INIT, Initiator/Handler Address: Points at the handler entry for the resource, i.e. the initiator for a device driver.

4) RDT.TERM, Terminator Handler Address: Points at an optional terminator, i.e. to do code conversions or CRC calculation in a device driver.

15.5 TASK DESCRIPTOR TABLE (TDT)

This table is a continuation of the RDT. It is used by the system to create a Task Control Block (TCB), which is then used to control a task. The information in the TDT is a short description of the characteristics of a task:



	Offset	Bytes	Type	Name	Description
1)	6	1	Byte	TDT.TYPE	Task type
2)	7	1	Byte	TDT.OPT	Task options
3)	8	2	Address	TDT.TLIM	Standard start address
4)	10	2	Integer	TDT.TLIM	Individual slice limit
5)	12	1	Byte	TDT.NNOD	Number of nodes
6)	13	1	Byte	TDT.NFCB	Number of FCB
7)	14	2	Integer	TDT.STK	Required stack size

Parameters

The parameters for the TDT are described in subsequent paragraphs.

1) TDT.TYPE, TYPE: This parameter describes the type of tasks as follows:

Bit	Value	Name	Description
0-1	-	-	Task status field
0	1	TCT.RES	Resident in memory
1	1	TCT.NAB	Non-abortable from other tasks

2) TDT.OPT, Options: The options parameter holds the task options.

Bit	Value	Name	Description
0-3	-	-	Task options field
0	1	TCO.DASG	Default assign allowed
1	1	TCO.NSTK	No stack check
2	1	TCO.EMSG	Error message print out by the system
3	1	TCO.RCOV	System recovery

3) TDT.SADR, Standard Start Address: This parameter contains the normal start address of the task.

4) TDT.TLIM, Individual Slice Limit: A task is given a time limit that it can use by this parameter if the time slice function is enabled. Zero means that the global time-slice limit should be used.

5) TDT.NNOD, Number of Nodes: This parameter specifies the number of nodes that should be allocated to the task. In general, the number of nodes required by a task is calculated from:

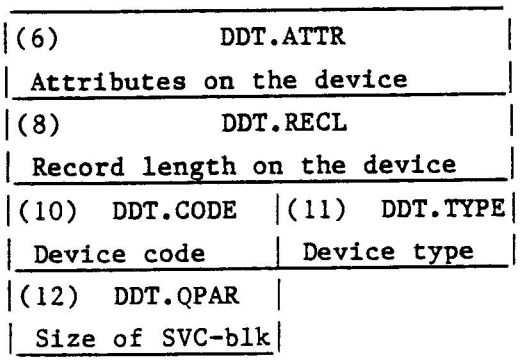
- The number of outstanding queued no-wait requests.
- The number of task devices owned by the task.
- The number of assignments to physical devices, not files.

6) TDT.NFCB, Number of FCB: The number of File Control Blocks (FCB) that should be allocated to a task is specified by this parameter. One FCB is used for each assigned file while one extra FCB is assigned to an element within a file.

7) TDT.STK, Required STack Size: This parameter specifies the required stack for the task. The byte allocation for the TDT is as follows:

15.6 DEVICE DESCRIPTOR TABLE (DDT)

Both real and task devices are specified by this table. The DDT is a continuation of RDT.



	Offset	Bytes	Type	Name	Description
1)	6	2	Byte	DDT.ATTR	Attributes on the device
2)	8	2	Integer	DDT.RECL	Record length on the device
3)	10	1	Byte	DDT.CODE	Device code
4)	11	1	Byte	DDT.TYPE	Device type
5)	12	1	Byte	DDT.QPAR	Size of SVC-blk

Parameters

The parameters for the DDT are described below:

1) DDT.ATTR, Attributes on the Device: This is a bit pattern which describes the attributes on the device.

It is a 16-bits long, and has a byte allocation defined as follows:

Bit	Name	Description
0	ATT.READ	Read
1	ATT.WRIT	Write
2	ATT.FASC	Formatted ASCII
3	ATT.STEC	Special formatting
4	ATR.RND	Random access
5	ATR.IACT	Interactive device echo of input

Bit	Name	Description
6-8	Reserved	
9	ATR.FR	Forward record
10	ATR.FF	Forward file
11	ATR.WF	Write file mark
12	ATR.BR	Backspace record
13	ATR.BF	Backspace file
14	ATR.RW	Rewind
15	ATR.ATTN	Attention

2) DDT.RECL, Record Length on the Device: This parameter specifies a record length. Zero means variable record length.

3) DDT.CODE, Device Code: The device code identifies a device among several having nearly the same type specification.

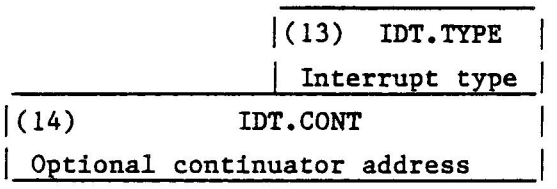
4) DDT.TYPE, Type: The type field gives the type of device. The a byte allocation for the device type is as follows:

Bit	Name	Description
0	DCT.CCB	Interrupt control block (ICB) present
1	DCT.DEDI	Dedicated interrupt service
2	Reserved	
3	Reserved	
4	DCT.TASK	Indicated a task device
5	DCT.DUAL	Dual DCB information

5) DDT.QPAR, Size of SVC-BLK: DDT.QPAR, specifies the number of bytes that should be copies from the parameter block to the DCB.

15.7 INTERRUPT DESCRIPTOR TABLE (IDT)

This table, which is a continuation of the DDT, holds a short description of the interrupt side of a device. The table is only necessary if specified by the function DCT.ICB, and is used to create an Interrupt Control Block (ICB). The parameter block for the IDT is shown below.



The parameter block for the IDT has the following structure:

	Offset	Bytes	Type	Name	Description
1)	13	1	Byte	IDT.TYPE	Interrupt type
2)	14	2	Address	IDT.CONT	Optional continuator address

Parameters

The parameters for the IDT are described below:

1) ID.TYPE, Interrupt Type: The type of interrupt is indicated as follows:

Bit	Value	Name	Description
0	1	ICT.CCB	Channel Control Block (CCB) present
1	1	ICT.NOIQ	Makes the ICB resident on the interrupt and time out chain

2) IDT.CONT, Optional Continuator Address: This parameter is used by the interrupt system as an address to the device driver continuator. This address is normally initiated or changed by the driver.

15.8 CHANNEL DESCRIPTOR TABLE (CDT)

This table, which is a continuation of the IDT, is used to create the Channel Control Block (CCB) for a device. This table is only required if specified by the function ICT.CCB in the IDT:

The parameter block for the CDT is as follows:

(16) CDT.TLIM
Time-out limit in chosen interval
(18) CDT.THND
Optional time-out handler address

The parameter block for the CDT has the following structure:

Offset	Bytes	Type	Name	Description
1) 16	2	Byte	CDT.TLIM	Time out limit in chosen interval
2) 18	2	Address	CDT.THND	Optional time out handler address

Parameters

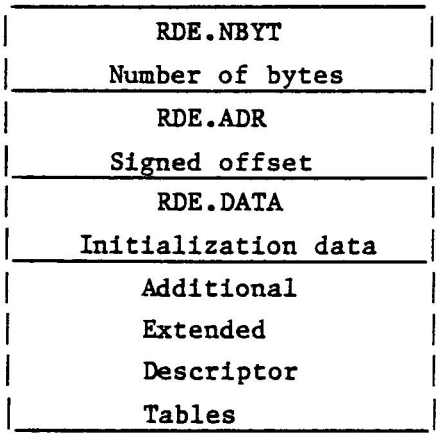
The parameter for the CDT are described below:

1) CDT.TLIM, Time-out Limit in Chosen Interval: This is the time-out limit of the device. The value is defined by the device time-out scan frequency, which is a system generation constant. Normally resolution is 100 ms.

2) CDT.THND, Optional Time-out Handler Address: This parameter holds the time-out handler address for the device. If not specified, the time-out situation is handled by the system.

15.9 EXTENDED DESCRIPTOR TABLE (EDT)

If an extension is specified in the RDT, the EDT must be added after the last descriptor table. This allows the programmer to both expand the control block, and initiate it with some data. If no initialization is required, the shortest EDT possible (one binary zero) must be added. Any number of EDT's may be concatenated to allow for initialization of arbitrary portions of the control block. The EDT is shown below:



The parameters for the CDT are listed and described below.

	Offset	Bytes	Type	Name	Description
1)	N+0	1	Integer	RDE.NBYT	Number
2)	N+1	1	Integer	RDE.ADR	Location of Initialization
3)	N+2	RDE.NBYT	Bytes	RDE.DATA	Initialization Data

1) RDE.NBYT, Number of Bytes: This parameter specifies the number of bytes that should be copied into the control block. A binary zero indicates that this is the last EDT and it will not contain RDE.ADR or RDE.DATA.

2) RDE.ADR, Signed Offset: This is a relative signed offset in the control block where to start to copy the data.

3) RDE.DATA, Initialization Data: This parameter contains the data to be copied to the control block. It must contain exactly RDE.NBYT bytes!

PART II

INPUT/OUTPUT MANAGEMENT

SECTION 16

CONSOLE MANAGEMENT

SECTION 16
CONSOLE MANAGEMENT

16.1 INTRODUCTION

The Console Management System is controlled by the user through a terminal device. The name of the logical terminal device is always CON for every user, and may be assigned to a task for ordinary I/O operations, just as any other device.

16.2 PROMPTING

When the terminal operator is expected to enter data at the terminal, a prompt is output. This prompt takes one of the following forms:

-	Command Request
(no prompt)	Data Request

The command request prompt is output whenever the system is ready to accept another command.

The data request prompt is output whenever a task is attempting to perform a read request to the terminal device. This request should be satisfied as soon as possible, since messages are held in abeyance until the data request is satisfied.

16.3 CONTROL CHARACTERS

Control characters are generated by holding down the control (CTRL) key and depressing the other key in the control sequence. For example, CTRL-X is entered with the Control key and the X key.

The control character conventions in effect for terminal devices are described below:

Keys	Function:
CTRL-X	Deletes a Line
CTRL-H	Deletes a Character (Backspace can also be used)
CRTL-=	End-Of-File Function
CTRL-A	Pauses Task (CTRL-A is enterd when the user wishes to communicate with the operating system rather than a task. Any input or output is suspended and the system responds with the command request prompt. It is then ready to accept another command.)
CTRL-C	Stops Task (Some commands and programs recognize a stop which aborts the terminal transfer and cancels the task execution.) Operating system commands such as CANCEL or TASK can be entered. The pause task is continued by entering a return key only. Any suspended I/O is resumed.

16.4 COMMAND HANDLING

The command is the basic unit of conversation between a terminal user and the system. A command directs the Command Management System to take a specific action. In general, a single command results in a single action being taken by the system.

A command consists of a mnemonic which normally describes the action the user wishes to take place, and arguments which provide the details necessary to perform the action.

Commands are accepted one line at a time with one command per line. A command may not spread over two or more lines. A command line is terminated by a carriage return.

Unknown commands

If an unknown command is entered, the system tries to load a program with the same name. If found, it will be started as a primary task, and the rest of the command line will be transferred as parameters to that task, otherwise the system responds with a load error.

Error Response

The Command Management system responds to a command error by typing out a message to the user indicating the type of error. In response to the error message the user must retype the entire command, correcting the error as necessary. The error messages are:

Error Message	Description
Load Error	Comamnd or program not found. Memory space not enough, or checksum error on a command or program file.
Size Error	Additional memory size improperly specified.
Seq Err	This message is given if the particular command cannot be accepted due to the state of the system. This occurs either when a command is executing and another command is entered, or when a primary task is executing and execution of a new program with the same name is requested.
Fd Err	Syntax error in a File Descriptor, or the type of the file is missing. It usually refers to the format of file names entered as parameters.

Error message	Meaning
Dev Err	Device not in syntax or not accessbile.
Id Err	Syntax error in a task name, or the task is not found.
Par Err	Parameter error, invalid or missing parameter.

SECTION 17

DEVICE DRIVER DESCRIPTION

SECTION 17
DEVICE DRIVER DESCRIPTION

17.1 INTRODUCTION

Each type of peripheral or task device has a control program called a device driver program. The driver handles all input and output for the device. It checks when transfer errors occur and reports these to the requestor.

Each driver has at least one point of entry, called an Initiator, and one exit point called a Terminator. Actual physical devices may also have a Continuator and/or a Time-Out/Cancel Handler. The driver is NOT allowed to use the secondary register set without saving it before it is used and restoring it after it is done.

Drivers that transfer data on a byte by byte basis normally use the Data Formatter to load and store the bytes.

17.2 DRIVER INITIATOR

The Driver Initiator is called from the Connection Handler and runs as a reentrant subroutine for the task issuing the I/O request. In general, the Initiator uses the information which has been stored in the DCB (by the Connection Handler) to prepare the information required to perform the requested function. This is often done through the Data Formatter.

After all processing has been completed, the Initiator starts the physical I/O process by causing an interrupt of the device requested. The Initiator then triggers the Connection Handler which returns control to the calling task on an I/O proceed call, or puts the calling task into I/O wait on an I/O wait call.

The following conditions are in effect when the driver is called.

- Executes in SMU mode.
- The SVC-block is copied into the DCB if specified.
- The initiator address of the Data Formatter is stored in DCB.FMTE.
- The ICB is linked into time-out and interrupt chain.

- The time-out counter CCB.TCNT is initialized from CCB.TLIM if the flag DCS.INT is cleared.
- Channel selected on physical device.
- Register X -> DCB.
- Register Y -> SVC-block.
- Register A := the function code without S1F.NW and S1F.PRO.

The following conditions are expected when the routing returns:

- Register X -> DCB.
- Register Y -> SVC-block.
- Register A := return status on completion.
- Carry flag.
- Reset means not complete.
- Set means complete.
- The flag DCS.INT in DCB.STAT must be set to enable interrupt polling and time-out checking if not complete.
- Carry flag set means complete.

17.3 DRIVER CONTINUATOR

When an interrupt is detected from a device the operating system causes control to pass to the Continuator. The Continuator is executed with all highter interrupt levels enabled. The actual I/O to the device is controlled by I/O instructions. On completion of all I/O requests the Continuator disables the interrupts from the device and returns to the System Interrupt Handler which adds the DCB to the system queue. The System Interrupt Handler always re-initializes the time out counter.

The following conditions are in effect when the continuator is called:

- Executes in IM.
- The time-out counter CCB.TCNT is initialized from CCB.TLIM.
- Channel selected on physical device.
- Register X -> DCB.
- On dedicated disabled interrupt, the driver is NOT allowed to enable the CPU, or use Register Y and Register pair BC!

The following conditions are expected when the routine returns:

- Register X -> DCB.
- Register A := return status on completion.
- Carry flag
- Reset, not complete.
- Set, complete.
- The flag DCS.INT cleared if no more interrupts are expected.

17.4 DRIVER TIME-OUT AND CANCEL

A hang-up error occurs when a device fails to generate an interrupt on an operation that was initiated. The time limit for this interrupt is computed by the Driver and stored in the DCB.TLIM, or is initiated at system generation time.

The error is detected by the Device Time-Out Manager, which decrements the time counter in the DCB. When the counter becomes zero and when no time-out is allowed (controlled by the flag DCS.TIME), the Time-Out Handler address, if specified, is scheduled. If a time-out is allowed, the continuator is called in the normal way.

The Time-Out Handler is also called when a request is canceled, and is responsible for the cancel checking!

If further more I/O must be initiated, the Time-Out Handler causes an interrupt on the device, often by re-entering the Initiator.

The following conditions are in effect when the routine is called:

- Executes in IM.
- Channel selected on physical device.
- Register X -> DCB.
- The flag DCS.INT in DCB.STAT is cleared.
- The flag DCS.TOUT in DCB.STAT is set.
- The flag RCS.CAN in RCB.STAT is set at cancel.

The following conditions are expected when the routine returns:

- Register X -> DCB.
- Register A := return status on completion.
- Carry flag
- Reset, not complete.
- Set, complete.
- The flag DCS.INT in DCB.STAT must be set to enable the checking.

17.5 DRIVER TERMINATOR

The Terminator is called from the Disconnection Handler as a result of a System Queue interrupt. The Terminator performs post-processing on any I/O request being terminated, such as code converting or CRC calculations. If additional I/O must be initiated, the Terminator causes an interrupt of the device, often by reentering the Initiator.

The following conditions are in effect when the routine is called:

- Executes in SMU if the calling task is in an I/O wait state, or in IM if no task is waiting.
- Channel selected on physical device.
- Register X -> DCB.
- Register Y -> SVC-block.

The following conditions are expected when the routine returns.

- Register X -> DCB.
- Register Y -> SVC-block.
- Register A := return status on completion.
- Carry flag
- Set, not complete.
- Set, complete.
- The flag ICT.NOIQ must be set if the ICB shall remain on the time-out and interrupt chain.

Example:

This is a simple example of an output driver that uses the Data Formatter. Note the critical instruction sequence when enabling the interface.

```
*
*
* DRIVER INITIATOR
* ----------------
*
* ON CALL:        X -> DCB
*                 Y -> SVC-BLOCK
*                 A := FUNCTION CODE MASKED
*
* ON RETURN:      X -> DCB
*                 Y -> SVC-BLOCK
*                CY := 0, NOT COMPLETE
*                CY := 1, COMPLETE
*                 A := RETURN STATUS
*
DRV.INIT  EQU   *
          CALL  DATA.FMT          THE DATA FORMATTER DOES THE CHECKING
          JTCS  WRONGFC           CAN'T HANDLE UNKNOWN FUNCTIONS !
          DECR  E                 EXAMINE FUNCTION REQUESTED.
          JTZS  WRONGFC           I DON'T SUPPORT READ.
          DECR  E
          JFZS  DONE              STANDARD FUNCTIONS, ACCEPT THEM.
          CALL  CHEK.PNT          WRITE, CHECKPOINT HERE...
*
*
* DRIVER CONTINUATOR
* ------------------
*
* ON CALL:        X -> DCB
*
* ON RETURN:      X -> DCB
```

```
*               CY := 0, NOT COMPLETE
*               CY := 1, COMPLETE
*                A := RETURN STATUS
*
DRV.CONT  EQU   *
          OR    A                 CLEAR CARRY BEFORE...
          CALL  DATA.FMT          ...LOAD THE NEXT BYTE...
          JTCS  COMPLETE
          OUT   DATA              ...THEN GIVE IT TO THE DEVICE.
          RET
*
*
* REQUEST COMPLETE
*
COMPLETE  EQU   *
          CALL  DATA.FMT          POSTPROCESS THROUGH DATA FORMATTER
*
DONE      EQU   *
          XR    A                 RETURN STATUS 0...
          OUT   C4                ...DISABLE INTERFACE INTERRUPT...
          RBT   DCS.INT,DCB.STAT(X)  ...DISABLE INTERRUPT POLLING...
          STC                     ...MARK COMPLETE...
          RET                     ...BACK TO SYSTEM.
*
*
* CALL THE DATA FORMATTER
*
DATA.FMT  EQU   *
          L     L,DCB.FMTE(X)     PICK UP THE ADDRESS TO THE CHECK-
          L     H,DCB.FMTE+1(X)   -POINTED DATA FORMATTER...
          JDR   HL                ...AND ENTER.
*
*
* CHECKPOINT AND WAIT FOR INTERRUPT
*
CHECK.PNT EQU   *
          POP   HL
          ST    L,ICB.CON(X)      SET UP INTERRUPT CONTINUATOR...
```

```
          ST    H,ICB.CON+1(X)
          MVI   80H,CCB.TM(X)     ...AND STATUS TEST MASK.
          LI    A,80H             LOAD INTERFACE CONTROL...
          DIS                     ...DISABLE CPU BEFORE MARKING...
          SBT   DCS.INT,DCB.STAT(X)  ...INTERRUPT POLLING ALLOWED...
          OUT   C4                ...AND ENABLE THE INTERFACE...
          ENI                     ...THEN ENABLE THE CPU.
          XR    4                 MARK NOT COMPLETE.
          RET                     BACK TO SYSTEM.
*
*
* WRONG FUNCTION CODE
*
WRONGFC   EQU   *
          LI    A,SOS.IFC         SET UP RETURN STATUS.
          STC                     MARK REQUEST COMPLETE.
          RET                     BACK TO SYSTEM.
```

SECTION 9
SVC 2 SUBFUNCTIONS

SECTION 9
SVC 2 SUBFUNCTIONS

9.1 INTRODUCTION

There are a number of service functions called subfunctions which are provided to the user. These are implemented by SVC 2.1 through SVC 2.12. These subfunctions are related to the tasks communication with the console operator, to memory allocation, to text processing, and to command processing.

The purpose of the function code for the subfunctions is generally to modify the conditions of the SVC call. The content of the function code is ignored by those subfunctions for which no function code has been defined. This means that no wait for completion and no unconditional proceed are supported. All subfunction requests require a Parameter Block accompanying the request.

The Parameter Blocks for the various SVC 2 subfunctions have the general form:

(0) S0.FC Function code	(1) S0.RS Return status
(2) S2.SNR Subfunction	(3) S2.PAR
(4) See SVC 2 subfunctions	

The parameter block for the SVC 2 subfunctions has the following structure:

	Offset	Bytes	Type	Mnemonic	Name
1)	0	1	Byte	S0.FC	Function Code
2)	1	1	Byte	S0.RS	Return Status
3)	2	1	Byte	S2.SNR	Subfunction
4)	3	1	Byte	S2.PAR	Parameter
	Total	4			

9.2 PARAMETERS

Descriptions of the parameters in the parameter block are given below:

1) S0.FC, Function Code

This parameter is determined by each subfunction.

2) S0.RS, Return Status

The Return Status is also determined by each subfunction. There is one common value, S2S.ISB, whose status code is 20 and which corresponds to an invalid subfunction number. S2S.ISB part of the Return Status Parameter of all SVC's. If a subfunction is used illegally, it is this status that will be returned.

3) S2.SNR, Subfunction

There are eight subfunctions available whose characteristics are as follows:

Subfunction	Status Code	Meaning
EV2.1MEM	1	Memory allocating.
EV2.2MSG	2	Log message.
EV2.3PFD	3	Pack file descriptor.
EV2.4PNU	4	Pack numeric data.
EV2.5UNP	5	Unpack binary number.
EV2.7DAT	7	Fetch or set date and time.
EV2.8CMD	8	Scan mnemonic table.
EV2.12OC	12	Open/close device.

4) S2.PAR, Other Data

The content of this field depends on each subfunction and is different for each one.

9.3 SVC 2.1 MEMORY HANDLING

SVC 2.1 is used to allocate and deallocate memory. The storage is allocated in system memory. Free space is allocated from the system table area (first 16K) and is very limited. Allocation is done on a first fit basis and the Operating System keeps a record of the size of the space that is actually allocated.

When releasing memory, only the location of the block to be released is required. Blocks of memory are released in the same fashion as they are accessed. For example, you cannot allocate a 2K byte block of memory and then release it in two 1K byte blocks. You must release it as a 2K byte block. Furthermore, deallocation must be done explicitly. The Operating System does support compaction.

Parameter Block

The Parameter Block for SVC2.1 is shown below.

(0) SO.FC Function code	(1) SO.RS Return status
(2) S2.SNR = 1	(3) S2.PAR Reserved
(4) S2.1ADR Memory address	
(6) S2.1SIZ Memory size	

The Parameter block for SVC2.1 has the following structure:

	Offset	Bytes	Type	Mnemonic	Name
1)	0	1	Byte	SO.FC	Function Code
2)	1	1	Byte	SO.RS	Return Status
3)	2	1	Byte	S2.SNR	Subfunction = 1
4)	3	1	Byte	S2.PAR	Reserved (not used) = 0
5)	4	2	Address	S2.1ADR	Memory Address
6)	6	2	Integer	S2.1SIZ	Memory Size
	Total	8			

Parameter

Descriptions of the parameter in the parameter block are given below.

1) S0.FC, Function Code

Memory Handling: The Memory Handling field is dependent upon the values of bits 0-2 as follows:

Bit	Value	Name	Description
0-2			Memory Handling Field
	1	S2F.1ALO	Allocate Memory
	2	S2F.1MAX	Reserved
	3	S2F.1REL	Release Memory
	4	S2F.1TCB	Allocated a Task Control Block (TCB). Only for internal use.
	5	S2F.1CAN	Remove a callers Task Control Block (TCB). Only for internal use.

2) S0.RS, Return Status: The Return Status Codes for SVC2.1 take on the values 21 and 22 as given below.

Function	Code	Meaning
S2S.1PAR	21	Illegal parameter.
S2S.1EOM	22	End of memory.

5) S2.1ADR, Memory Address: Contains the memory address at deallocation and returns the memory address at allocation.

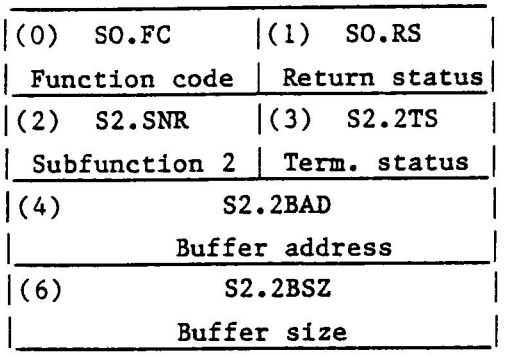
6) S1.ISIZ, Memory Size: Specifies the memory size to be allocated in bytes.

9.4 SVC 2.2 LOG MESSAGE

SVC 2.2 is used to log a message on the terminal device or system log device irrespective of the logical unit assignments in force at the time of the request.

Parameter Block

The Parameter Block for SVC 2.2 is shown below.



The Parameter Block for SVC 2.2 has the following structure:

	Offset	Bytes	Type	Mnemonic	Name
1)	0	1	Byte	SO.FC	Function Code
2)	1	1	Byte	SO.RS	Return Status
3)	2	1	Byte	S2.SNR	Subfunction = 2
4)	3	1	Byte	S2.2TS	Reserved = 0
5)	4	2	Address	S2.2BAD	Buffer Address
6)	6	2	Integer	S2.2BSZ	Buffer Size
	Total	8			

Parameters

Descriptions of the parameters in the parameters block are given below.

1) SO.FC, Function Code: Refer to the SVC 1 Function Code parameters regarding data formatting.

2) SO.RS, Return Status: There is no return status.

3) Reserved.

4) S2.2BAD, Buffer Address: The address of the buffer to write on the system console.

6) S2.2BSZ, Buffer Size: This parameter specifies the number of bytes to write.

Example

```
        Log the message 'Records Copied' onto the console.
S22ANTR     DB      0
*           DB      0
            DB      EV2.2MSG
            DB      0
            DA      ANTRTXT
            DA      ANTRSIZ
*
ANTRTXT     DB      '   Records  Copied.
ANTRSIZ     EQU     *-ANTRTXT
SVC                 2,S22ANTR
```

9.5 SVC 2.3 PACK FILE DESCRIPTOR

SVC 2.3 allows the user to process a File Description in standard Monroe syntax. The scan proceeds until it has satisfactorily processed each field in the File Descriptors syntax. Headings and spaces are ignored. If the scan finds illegal characters, a syntax error is returned and the scan terminates. Note that some kind of termination character must exist if the string size is not specified.

Parameter Block

The Parameter Block for SVC 2.3 is shown below.

(0) S0.FC	(1) S0.RS	(2) S2.SNR=3	(3) S2.3TS
Function code	Return status		Term. status
(4) S2.3ADR		(6) S2.3BUF	
ASCII-string address		Address of receiving area	
(8) S2.3PNT		(10) S2.3CNT	
Terminating string address		String size	

The Parameter Block for SVC 2.3 has the following structure:

	Offset	Bytes	Type	Mnemonic	Name
1)	0	1	Byte	S0.FC	Function Code
2)	1	1	Byte	S0.RS	Return Status
3)	2	1	Byte	S2.SNR	Subfunction = 3
4)	3	1	Byte	S2.3TS	Termination Status
5)	4	2	Address	S2.3ADR	ASCII-String Address
6)	6	2	Address	S2.3BUF	Address of Receiving Area
7)	8	2	Address	S2.3PNT	Terminating String Address
8)	10	2	Integer	S2.3CNT	String Size
	Total	12			

Parameters

Descriptions of the parameter in the parameter block are given below.

1) S0.FC, Function Code: The function code byte allocation is given as follows:

Bit	Value	Name	Description
0	1	S2F.3FN	Unpack as filename if not specified.
1	2	S2F.3KEP	Keep non-modified fields.
2	4	S2F.3CNT	String size specified.
3	8	S2F.3PMO	Pack modifier.

2) S0.RS, Return Status: The Return Status parameters are as follows:

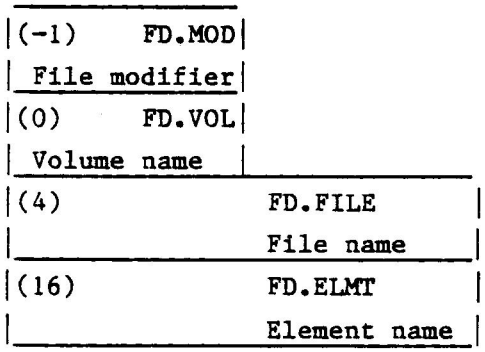
Function	Code	Meaning
S2S.3IFD	21	Invalid file descriptor, syntax error.

4) S2.3TS, Termination Status: The termination status byte allocation is given in the following table:

Bit	Value	Name	Description
0	1	S2T.3NEL	Element name not found.
1	2	S2T.3NFN	Filename not found.
2	4	S2T.3NVO	Volume name not found.
3	8	S2T.3NMO	Modifier not found. This status is only set if the pack modifier field S2F.3PMO is requested.

5) S2.3ADR, String Address: Is a pointer to a string that contains the file descriptor to be packed.

6) S2.3BUF, Receiving Area: This is a pointer to a 29-byte area. Note that the modifier field is on the negative side of the area, and must only be present if the function S2F.3PMO is requested. The following parameters are then included in the Parameter Block:



The modifier field has the following structure:

Offset	Bytes	Type	Mnemonic	Name
-1	1	Byte	FD.MOD	File Modifier
0	1	Byte	FD.VOL	Volume Name
4	2	Address	FD.FILE	File Name
16	2	Byte	FD.ELMT	Element Name

7) S2.3PNT, Terminating String Address: This field is returned pointing to the first byte that is not part of the file descriptor.

8) S2.3CNT, String Size: This is an optional field, specifying the string length. The lenght of this string can be limited by setting the S2F.3CNT-bit in SO.FC field, and give the size in S2.3CNT-field. A string length of zero means that the source string is terminated with 0.

9.6 SVC 2.4 PACK NUMERIC DATA

SVC 2.4 translates ASCII hexadecimal, decimal, or octal character strings into binary 8, 16, 24, or 32 bit numbers. Leading spaces are ignored and the conversion continues until a character not conforming to the base is found.

Parameter Block

The parameter block for SVC2.4 is shown below.

(0) S0.FC	(1) S0.RS	(2) S2.SNR = 4	(3) S2.4SIZE
Function code	Return status		Size
(4) S2.4ADR		(6) S2.4PNT	
String address		Updated string address	
(8) S2.4RES			
Result			

The parameter block for SVC 2.4 has the following structure:

	Offset	Bytes	Type	Mnemonic	Name
1)	0	1	Byte	S0.FC	Function Code
2)	1	1	Byte	S0.RS	Return Status
3)	2	1	Byte	S2.SNR	Subfunction = 4
4)	3	1	Integer	S2.4SIZE	Size
5)	4	2	Address	S2.4ADR	String Address
6)	6	2	Address	S2.4PNT	Updated String Address
7)	8	4	Byte	S2.4RES	Result
	Total	8			

Parameters

Descriptions of the parameters found in the parameter block are given below:

1) SO.FC, Function Code: The function code byte allocation is as follows:

Bit	Value	Name	Description
0-2	-	-	Conversion Base.
	0	S2F.4DEC	Decimal.
	1	S2F.4OCT	Octal.
	2	S2F.4HEX	Hexadecimal.
3			Sign Handling.
	0	S2F.4SGN	No sign input allowed.
	1	S2F.4SGN	Input may be signed.
4			Destination.
	0	2SF.4IND	In parameter block.
	1	2SF.4IND	Address Specified.

2) SO.RS, Return Status: The Return Status codes can take on the values of 21 and 22 as shown below.

Function	Code	Meaning
S2S.4OFL	21	Overflow.
S2S.4NCV	22	Nothing converted.

3) S2.4SIZE, Size: This parameter describes the size of the binary result. The user can auto-increment the result pointer using this parameter. The bit contents is:

Bit	Value	Name	Description
4-6	-	-	Size of result field.
		S2Z.4BIN	Contains the number of bytes in the result field.
7	-	-	Result poitner field.
	0	S2Z.$INC	Do not auto increment the result pointer.
	1	S2Z.4INC	Auto increment the result pointer.

Note that the size of the buffer through which the translation is invoked must be specified in advance. S2Z.4BIN then indicates the size of the binary result. If for example, you have characters in an array, that must be translated into binary S2Z.4BIN can be used to position the buffer into segments of some fixed memory length. Hence, if S2Z.4BIN is set at K bytes then a 1 in bit 7 for S2Z.4INC means that each time the result pointer is incremented it will do so in K byte segments of memory. During the actual data translation phase this removes an extra program step since you do not have the extra statement incrementing the result pointer after the previous array element has been buffered.

5) S2.4ADR, String Address: This is a poninter to the first character of the ASCII string to be converted.

6) S2.4RES, Result: The result is placed either in this field, or at the address specified by this field.

7) S2.4PNT, Updated String Address: This is the updated string pointer at return, and it is pointing at the first byte in the string that was not convereted.

Ex. 1

This example converts ASCII decimal to a binary number.

```
SVC24     DB      S2F.4DEC
STAT      DB      0
SUB       DB      EV2.4PNU
SIZE      DB      STRLEN
STRAD     DA      STRDATA
STRRS     DA      0
RESBIN    DMA     2,0
STRDATA   DB      '1234567'
STRLEN    EQU     *-STRADATA
SVC               2,SVC2.4
```

9.7 SVC 2.5 UNPACK BINARY NUMBER

SVC 2.5 translates an 8, 16, 24 or 32 bit number into ASCII, hexadecimal, decimal, or octal format.

Parameter Block

The parameter block for SVC2.5 is shown below.

(0) SO.FC	(1) SO.RS	(2) S2.SNR = 5	(3) S2.5SIZE
Function code	Return status		Size
(4) S2.5ADR		(6) S2.5PNT	
Destination address		Updated string address	
(8) S2.5VAL			
Source			

The Paramter Block for SVC 2.5 has the following structure:

	Offset	Bytes	Type	Mnemonic	Name
1)	0	1	Byte	SO.FC	Function Code
2)	1	1	Byte	SO.RS	Return STatus
3)	2	1	Byte	S2.SNR	Subfunction = 5
4)	3	1	Integer	S2.5SIZE	Size
5)	4	2	Address	S2.5ADR	Destination Address
6)	6	2	Address	S2.5PNT	Updated String Address
7)	8	4	Byte	S2.5VAL	Source
	Total	12			

Parameters

Descriptions of the parameters are given below.

1) SO.FC, Function Code: The function code byte allocation is as follows:

Bit	Value	Name	Description
0-1			Base field.
	0	S2F.5DEC	Decimal.
	1	S2F.5OCT	Octal.
	2	S2F.5HEX	Hexadecimal
2			Sign field.
	0	S2F.5SGN	Unsigned conversion.
	1	S2F.5SGN	Signed conversion
3			Source description
	0	S2F.5IND	Number in parameter block.
	1	S2F.5IND	Address specified.
4			Space flag.
	0	S2F.5SP	Leading zeros.
	1	S2F.5SP	Leading spaces.
5			Field Justification.
	0	S2F.5LFT	Right Justify.
	1	S2F.5LFT	Left Justify.

Note that when converting a number to hexadecimal, decimal, or octal the base must be specified.

2) SO.RS, Return Status: There is no return status.

3) S2.5SIZE, Size: Describes the size of both the binary and ASCII fields, and allows you the possibility of auto-incrementing the binary pointer. If the number to be converted exceeds the buffer length, most of the significant bytes are lost. If suppression of leading zeros is requested, the number is stored in the buffer, and the remaining characters, if any, are filled with spaces. If the number is to be left justified, only the number of bytes required for the number will be used,

and S2.5PNT will point at the next position after the number. The byte allocation size is as follows:

Bit	Value	Name	Descripton
0-3			Bytes in ASCII.
	-	S2Z.5ASC	Contains the number of bytes in an ASCII character string.
4-6	-	S2Z.5BIN	Contains the number of bytes in a binary number.
7			Auto increment.
	0	S2Z.5INC	Do not auto increment binary pointer.
	1	S2Z.5INC	Auto increment binary pointer.

5) S2.5ADR, Destination Address: This is a pointer to the first location of a buffer in memory where the converted number is to be stored. This buffer must be in a writable logical segment.

6) S2.5PNT, Updated Destination Address: This is the updated buffer address pointer at return, and it is pointing at the first byte in the buffer after the converted number.

7) S2.5VAL, Source: The binary number is placed either in this field, or at the address specified by this field.

Ex.

```
S25INERR  DB    S2F.5DEC + S2F.5SP
          DB    0
          DB    EV2.5UNP
          DB    10H+3
          DA    INERR
          DA    0
          DA    0
          DA    0
SVC 2,S25INERR
```

9.8 SVC 2.7 FETCH/SET OR DATE/TIME

SVC2.7 is used either to interrogate the time-slice value or to fetch and set the current time of day in the operating system.

Parameter Block

The parameter block for SVC2.7 is shown below.

(0) SO.FC Function code	(1) SO.RS Return status
(2) S2.SNR = 7	(3) S2.PAR Reserved
(4) S2.7BUF Buffer address	

The Parameter Block for SVC2.7 has the following structure:

	Offset	Bytes	Type	Mnemonic	Name
1)	0	1	Byte	SO.FC	Function Code
2)	1	1	Byte	SO.RS	Return Status
3)	2	1	Byte	S2.SNR	Subfunction = 7
4)	3	1	Byte	S2.PAR	Reserved
5)	4	2	Address or Integer	S2.7BUF	Byte for address or time slice value
	Total	6			

Parameters

The parameters found in the parameter block are described below:

1) S0.FC, Function Code: the byte allocation for the function code is as follows (S2.PAR is not used).

Bit	Value	Name	Description
0-1			Fetch 1 set field
	0	Illegal	-
	1	S2F.7GET	Fetch function
	2	S2F.7SET	Set function
	3	Illegal	-
2-3			Slice/date/time field
	0	S2F.7SLC	Slice handling
	1	S2F.7DAT	Date handling
	2	S2F.7TIM	Time handling
	3	(S2F.7DAT	Date and time handling.
		+ S2F.7TIM)	The buffer has the date followed by the time.
4-5			Data field
	0	S2F.7ASC	ASCII data; slice handling only.
	1	S2F.7BIN	Binary data, time and/or date handling only.

Note in all of the above fields there is a 0 in bit 5.

2) S0.RS, Return Status: The following return status codes are in effect.

Function	Code	Meaning
S2S.7DAT	21	Invalid date.
S2S.7TIM	22	Invalid time.

5) S2.7BUF, Buffer Address: This field 1) holds the value at slice handling. 2) Contains the address to a buffer within the user's program that either receives or sends the values at date and/or time handling.

SECTION 18

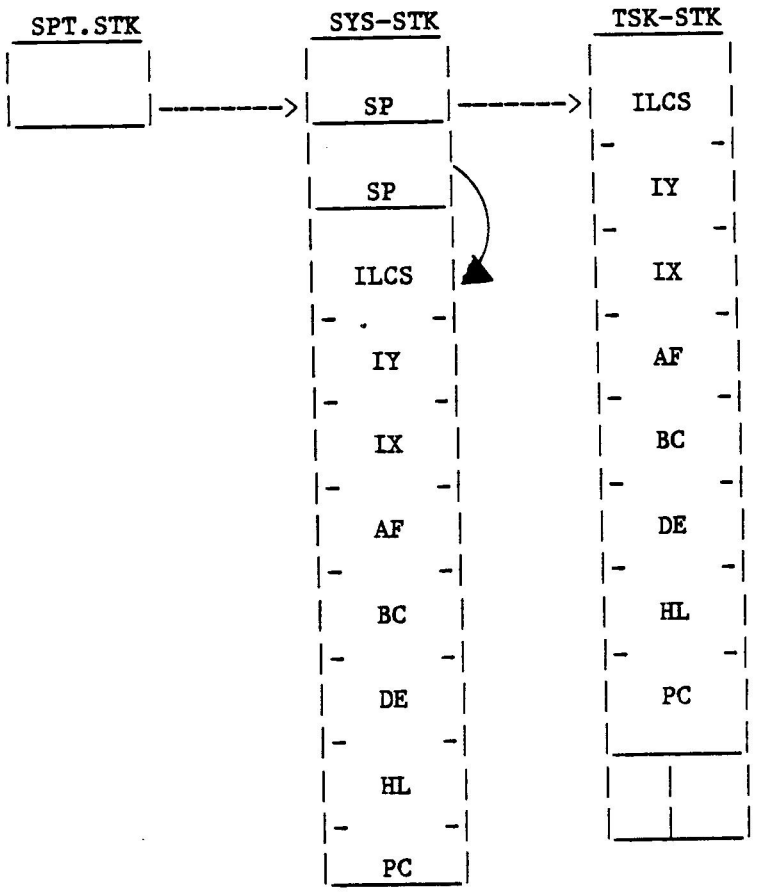
INTERRUPT STRUCTURES

SECTION 18

INTERRUPT STRUCTURES

18.1 STACK FOR SYSTEM ROUTINES

The System Pointer, System Stacks, and Task Stacks have the following structure.



SPT.STK

Contains the address to the system stack that should be used during an interrupt. It is located in the system area.

SYS-STK

Is the structure of the system stack when the system is executing in Interrupt Mode. It is located in the system area (lower 16KB).

TSK-STK

The structure of a task stack when a task has been interrupted. It is located in a tasks impure (B-segment) area.

Stack Table for System Routines (In the system stack area)

	Offset	Size	Type	Name	Description
1)	0	2	Word	SSP.RET	Interrupt Handler Address
2)	2	2	Word	SSP.BSEG	Task BSEG Base
3)	4	2	Word	SSP.SP	Task SP
	Total	6			

Stack Table for Interrupt (On the User stack)

	Offset	Size	Type	Name	
1)	0	2	Word	USP.ILRS	
2)	2	1	Byte	USP.MLIM	
3)	3	1	Byte	USP.ASEG	
4)	4	2	Word	USP.IY	
5)	6	2	Word	USP.IX	Description
6)	8	2	Word	USP.AF	Previous level & channel
7)	10	2	Word	USP.BC	User mode and limit
8)	12	2	Word	USP.DE	User a segment
9)	14	2	Word	USP.HL	CPU register IY
10)	16	2	Word	USP.PC	
11)	18	2	Word	USP.SP	
	Total	20			

Stack Table for Suspension (On the user stack)

	Offset	Size	Type	Name
1)	0	2	Word	TSP.AF1
2)	2	2	Word	TSP.BC1
3)	4	2	Word	TSP.DE1
4)	6	2	Word	TSP.HL1
5)	8	20	-	Same as for interrupt stack above
	Total	29		

Stack Table for Kernel Mode

	Offset	Size	Type	Name
1)	0	2	Word	KSP.HL
2)	2	2	Word	KSP.PC
3)	4	20	-	Same as for interrupt stack
	Total	24		

Stack Table for Queue

	Offset	Size	Type	Name	Description
1)	0	2	Address	QUE.LNK*	Next item in the list
2)	2	1	Byte	QUE.PRI*	Priority (seating item)
3)	3	1	Byte	QUE.NR*	Data
	Total	4			

Stack Table for Node

	Offset	Size	Type	Name	Description
1)	0	2	Address	NOD.LNK	Next node in list
2)	2	1	Byte	NOD.CPRI*	Priority
3)	3	1	Byte	NOD.CTCB*	TCB number
4)	4	1	Byte	NOD.QFC*	Request function
5)	5	1	Byte	NOD.SEG*	SVC segment value
6)	6	2	Address	NOD.SVC*	SVC parameter address
7)	8	1	Byte	NOD.RNR*	Resource number
8)	9	1	Byte	NOD.EVNT*	Event number
9)	10	2	Add/integer	NOD.RCB* (.TIME*)	RCB add or timer value
10)	12	2	Address	NOD.AQLK*	Assign link
11)	14	2	Integer	NOD.OWN*	Reserved
	Bytes	16			

18.2 LOCATING TASKS IN MEMORY

To locate tasks within operating system memory, load the task you wish to locate and get its task number via the TASK command which lists the tasks, their ID and numbers. (c.f. MONROE UTILITY PROGRAMS PROGRAMMERS REFERENCE MANUAL). Second, load the BIAS register with 0 (BIAS 0). The EXAMINE and MODIFY can now look at the operating system in the first 16K bytes of memory.

Initially you should look at the free space table used by the operating system. The table is located at 3F80 (hex). It contains one byte for each 1K bytes of memory (i.e. 128 bytes in the table for the 128K bytes of Monty memory). Each byte contains the task number for which the corresponding 1K byte of memory is allocated. The operating system's task number is FF hex and free space is 0.

The next step is to calculate the base address of the program which is 1024 (decimal) times the offset of the first byte in the space table which contains the task number in question. The result can then be placed into the BIAS register via the BIAS command (remember that BIAS requires a hex value). The program can then be Examined or Modified.

One helpful trick is to subtract the corresponding program offset from the value given to the BIAS command so that addresses used with the Examine and Modify commands and the program will be the same. For example, programs usually start at E000 hex. If the memory allocation begins at 1E000 hex then a BIAS 10000 command would set up the access such that Examine E000 would list the first byte of the program located at logical address E000 hex and physical address 1E000 hex. Note that the lower 16K allocated to the operating system is not accessible with low addresses like 0 because the program memory mapping scheme is not in effect.

SECTION 19

DATA STRUCTURES

SECTION 19
DATA STRUCTURES

19.1 INTRODUCTION

This section contains detailed descriptions of memory management, segmented code files, various formats for the system control blocks and formats for all table entries.

The central resource of the operating system is the System Pointer Table (SPT). This table contains general operating system information and pointers to the lists of resources in the system. The elements of these lists are described after the SPT. Every public resource must be defined at a numeric level and optionally defined at a symbolic level. This is done by a Resource Reference Table (RRT) at the numeric level and a Resource Mnemonic Table (RMT) at the symbolic level. These tables are then linked together. In addition, an exclusive resource must also have a Resource Control Block (RCB) which controls both the access to and establishes the queue into that resource.

19.2 MEMORY MANAGEMENT

In order to gain an understanding of the various System Control Blocks and Table Entries it is necessary to review how memory is managed by the Monroe Operating System. This is summarized by the diagram in Figure 19-1.

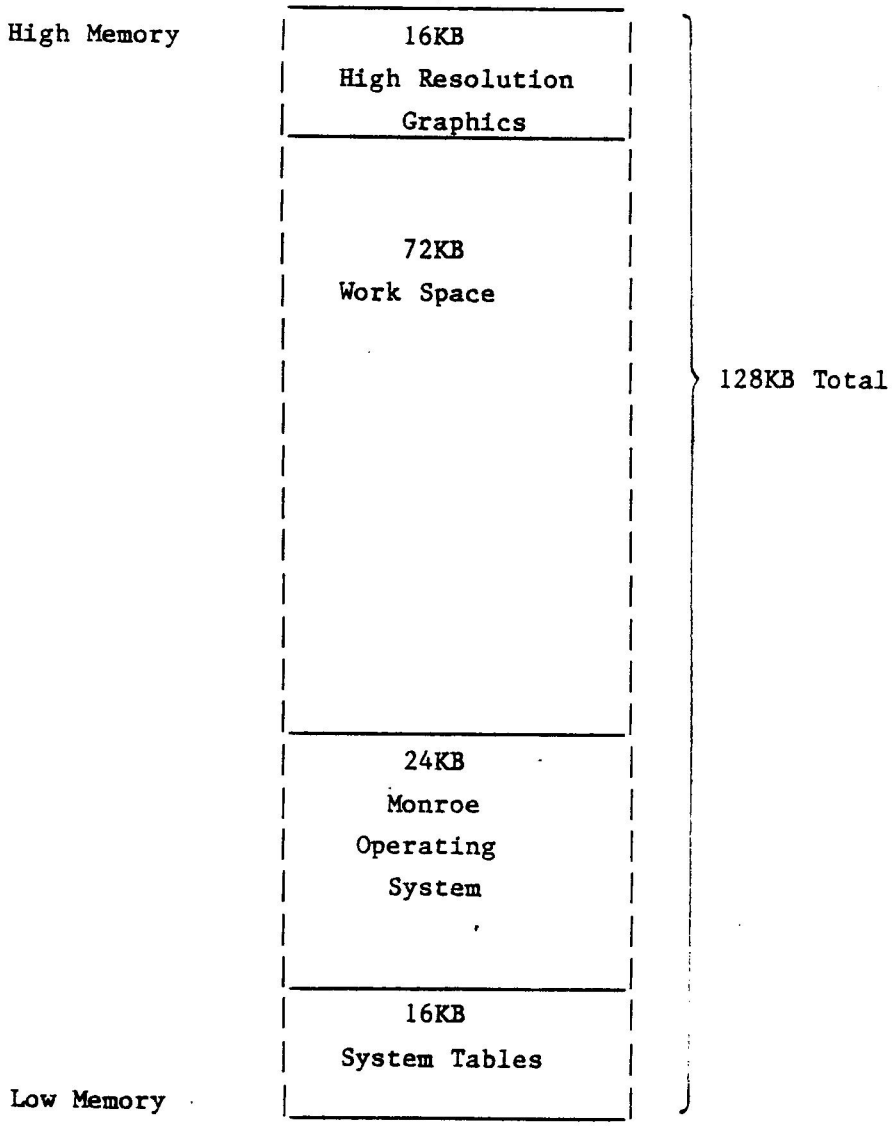


Figure 19-1. Physical Memory

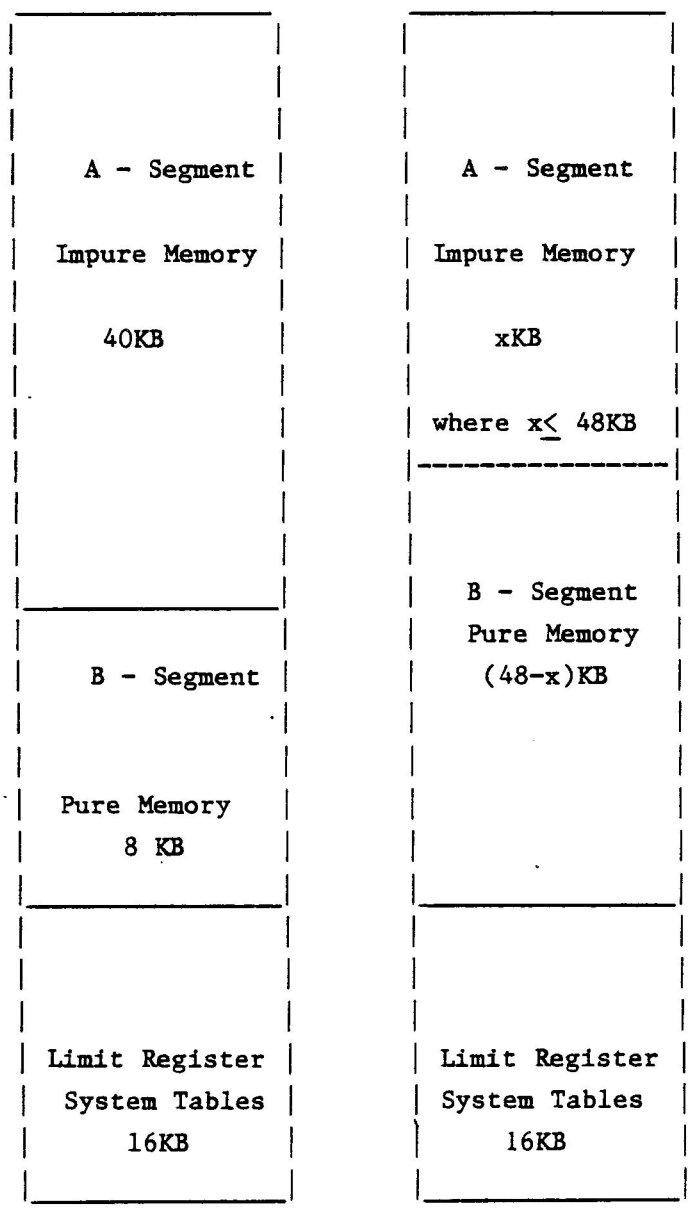
Physical Memory

The first 16K bytes of physical memory are reserved for the System Tables. The next 24K bytes are reserved for the Monroe Operating System. At the top (located in High Memory) there are 16K bytes allocated for High Resolution Graphics. The middle segment, 72K bytes long, is Work Space which can be partitioned in any way depending upon the program configuration being executed. This work space can be further subdivided into pure and impure code. there is a segment, which can be at most 49K bytes, called impure memory and another segment, which can be at most 40K bytes, called pure memory. These two 40K blocks are essentially two contiguous segments located in a 72K work space.

Logical Memory

When a program starts up it needs a certain amount of physical memory space to execute. This space may be 0 bytes; nevertheless, normally a program will need a certain fixed number of bytes of pure and impure memory in order to run. This space is allocated by the Operating System out of the work space consisting of 72K bytes. In addition, the Operating System keeps a free spare table indicating what can be used and what cannot be used.

When the user loads and executes a program all of the operating system tables set their pointers to the Z-80 Logical Memory Space. The bottom 16K bytes of this Logical Memory Space is reserved for the System Tables. The remaining 48K bytes are used to access both the pure and impure memory space in physical memory. Normally this is done in segments of 40K and 8K bytes, however this space can be partitioned in any way so long as the total number of bytes is 48. This is done by two registers; an A-segment register, and a B-segment register-both of which point to the work space. A limit register, in logical memory, indicates where the break point occurs (See Figure 19-2). This configuration, is called a Task Space, or Task Addressing Space.



(a) Normal Segmentation (b) Optional Segmentation

Figure 19-2. Task Addressing Space

Memory Mapping

Within each system table there is an address which is associated with a segment value that is one byte long. This pair, consisting of the address and the segment value, is what accesses the actual physical memory space of the operating system. The segment value is loaded into the appropriate segment register (usually the B register segment) and then the address accesses the actual data.

Therefore, if you were to map this Logical Memory Space or Task Space onto physical memory you might do it in the manner suggested by Figure 19-3. In this figure 40K bytes are allocated for impure code, 8K bytes for pure code, and 16K bytes for the system tables. Note that the 40K and 8K bytes are mapped onto different areas of the 72K physical work space while the 16K bytes for the system tables are mapped onto the 16K bytes at the low end of physical memory. Note also that the logical memory space of the Z-80 is constructed so that even though it is only 64K bytes long it can freely access 128K bytes of physical memory; that is, it occupies a physical memory space of 64K bytes and a logical memory space of 128K bytes.

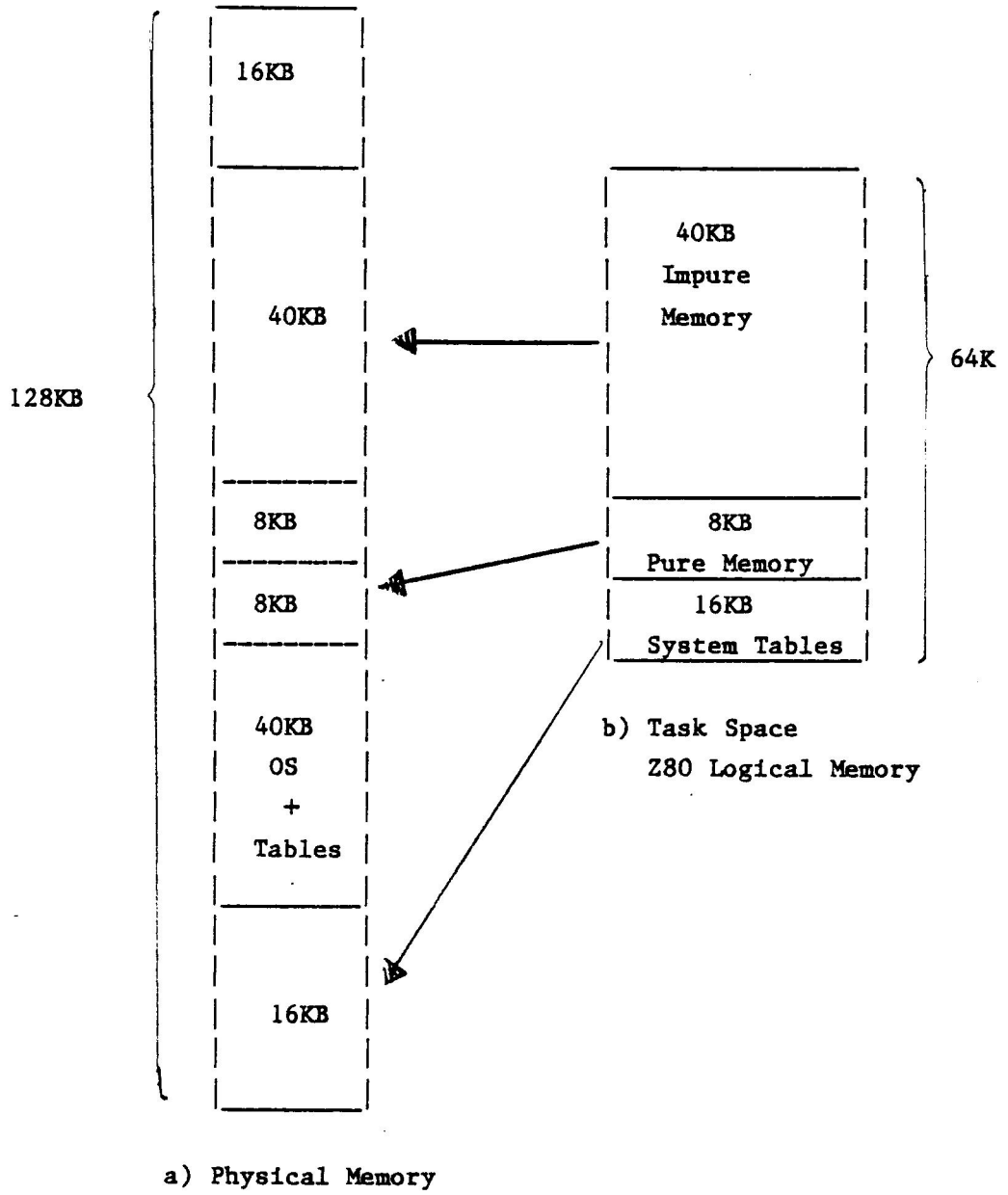


Figure 19-3 Memory Mapping (Logical to Physical)

19.3 TWO SEGMENT CODE FILES AND PROGRAMS OVER 40K

Executable code resides in a Monroe Operating System task file. An example of a task file is the operation system. A task file is placed into memory by the Monroe OPERATING SYTEM loader which is capable of loading either absolute code or elocatable code. Loading of relocatable code requires the additional step of converting the code to an absolute format before loading it into memory.

Memory Partitions

The memory space for a task is divided into three segments: the system area, the A segment (pure segment), and the B segment (impure segment). The system area is always 16K and is mapped into the lowest 16K of physical memory (128K). The A segment and B segment are of variable size (granularity is 8K) and must total 48K. The A segment always begins at 4000 hex and the B segment always ends at FFFF hex. The boundary is programmable however, by convention, the A segment is 8K and the B segment is 40K.

The A and B segments can be mapped into any place in memory (on 8K boundaries) with the logical space overlapping the physical space. A and B can even map the same space.

The system area contains operating system code and tables and should not be used to load any program but the operating system.

The A segment contains pure code which means that only program code is contained in the segment. It also means that the code can be shared if two programs can use the same code as with Monroe BASIC.

The A segment can logically address 8K (by system convention), however, the loader can put up to a 40K segment into memory. This means that up to 32K could not be accessed, but, by good design, the entire 40K can be accessed in 8K segments by an operating system overlay method discussed later. Therefore, a program can have up to 40K of program code and 40K of additional code and data. Note: to use the A segment requires a special procedure also described later.

Table 19-1. Loader Information Block (LIB)

Note: All integers are MSB (low offset) to LSB (high offset)

	Offset	Bytes	Type	Description
1)	0	2	Word	LIB Designation, must be 200 Hex
2)	2	4	String	Date created (packed BCD) (optional)
3)	6	4	String	Time created (packed BCD) (optional)
4)	10	2	Integer	Version number (optional)
5)	12	4	Long Int.	Execution start address (absolute code only)
6)	16	4	Long Int.	Stack size in bytes
7)	20	2	Word	Code type (see below)
8)	22	1	Byte	Task type (see SVC8)
9)	23	1	Byte	Task Options (see SVC8)
10)	24	2	-	Reserved
11)	26	1	Byte	Task priority (128=Default)
12)	27	1	-	Reserved
13)	28	4	String	Task name (for pure code only)
14)	32	96	-	Reserved
15)	128	1	Byte	Code checksum (absolute code only)
16)	129	1	-	Reserved
17)	130	2	Address	Load address (absolute code only)
18)	132	2	Integer	Code size in bytes
19)	134	21	-	Reserved
20)	255	1	Byte	LIB checksum
		256 Bytes		

7) Code Type Word: This byte contains the following:

Bit	Size	Description
0	1	Absolute code (0=Relocatable)
1	1	Reserved
2	1	Segment A-Pure Code (0=Segment B)
3	13	Reserved

The optional items should be zero if not included. The execution start address should be under 10000 hex and it is applicable to absolute code only. Relocatable code includes the start address.

The stack size should also be under 10000 hex and it indicates the amount additional memory required by the program in addition to code space. The total amount of allocated space is the code size plus the stack size. Note that the code size is the amount to allocate. It also specifies the amount in the file if it is in absolute format only.

The task priority and options are mentioned in the system documentation under SVC8.

The task name is used only with the pure segment. The impure segment derives its name from the initializing program, such as the operating system.

Note: If a pure segment is to be loaded and one copy already exists in memory (i.e. their task names are the same), then the second copy will not be loaded and the memory copy will be used.

Note: If a pure and impure segment is loaded, there will be two tasks created but only one will be executed. This is due to the allocation scheme.

Checksum Computation: All data bytes plus the checksum, when added with carry, must result in an 8 bit zero value. The initial state of the carry is zero. The addition should be modul0 256 and carry should be the result of the add divided (integer) by 256 (i.e., the overflow).

File Format: The format of a task file is:

	Sectors	
1)	1	Impure LIB (segment B)
2)	N	Impure code
3)	1	Pure LIB (Segment A)
4)	N	Pure code

The impure code section is omitted if it is not required. (i.e. it is only a non-initialized data area). The pure segment and pure code are also optional, however, if you have only an impure segment the impure code section should contain executable code.

Also, if the impure segment is omitted then the impure LIB should indicate a relocatable format.

Special Loader Information for Pure Segments

Impure LIB

Execution start address must point to the absolute start address of the program which may be segment A or B. The impure LIB also specifies the task priority, type, and options.

Pure LIB

Execution start address and stack size should be zero.

Memory Allocation Procedure

The impure segment is allocated in logical memory from FFFF hex down to FFFF hex minus the size (code plus stack) of the impure segment. Unallocated memory should not be used.

The pure segment is always located at 4000 hex to 5FFF hex in the logical space. If the segment is physically larger, then the first 8K portion is made immediately accessible. The remaining portion will be adjacent in physical memory but must be accessed during procedures described later.

Program Transfer Procedure

The A segment (pure code) is 8K in the logical space but up to 40K in the physical space. This means that only one 8K section of the segment is accessible to a program at any one time.

Since it is assumed that only code will reside in the A segment and it will be executed, the routine available from the operating system will jump from one logical A segment to another. The location of the system routine is 052 hex and it will be called TRANSFER in the examples.

TRANSFER takes a segment value and an absolute branch address as parameters with control being passed to the new location. The segment value is passed in the A register and is eight (8) times the logical block index where the first block is zero and the last is five (for 40K). Therefore, 0,8, and 16 correspond to blocks 0, 1, and 2 (note that the partition boundaries are 0K, 8K and 16K).

The routine address in the new segment is placed on top of the stack, and a branch to the TRANSFER routine is made. The routine swaps the current segment number with the new segment number in the A register and then branches to the new routine. The DE and HL registers in the Z80 are lost. A table of the state change is:

	Before	After
Register A	New Segment	Old Segment
Register DE	Data	Garbage
Register HL	Data	Garbage
Top of Stack	Branch Address	(Previous value)
Segment Register	Old Segment	New Segment
Flags	Data	Garbage
Program Counter	TRANSFER	Branch Address

For branches, the old segment index can be ignored, however, for procedure calls it must be saved along with the return PC value, but note that this must be explicitly by your program. This can be done as a two step process: (1) call a local routine that branches to TRANSFER and (2) have the routine called in the new segment save the A register (old segment value) and restore it when it is done. The routine terminates by branching to TRANSFER. (Note: this requires an additional routine in the calling segment.)

The second method of doing a long call involves additional routines in both the calling segment and routine segment. However, the called routine is the same as a normal routine (i.e. it ends with a return not a jump to TRANSFER). This method costs a few additional bytes of memory for only one usage, but it can save memory if there are a number of routines.

Note: The stack cannot be in segment A.

Examples:

1) Jump only: Jump to routine in segment 1.

In Old Segment:

```
          LI      A,8             New segment 1
          LA      HL, ROUTINE     PUSH routine address
          PUSH    HL
          JMP     TRANSFER        TRANSFER to new segment
```

In New Segment:

```
ROUTINE   =       *               Normal Routine
```

2) CALL Method 1: Call Routine in Segment 2

In Old Segment:

```
          LI      A,16            New segment
          LA      HL, ROUTINE     Routine in new segment
          CALL    LONG CALL

LONGCALL  =       *
          PUSH    HL              Return address and
          JMP     TRANSFER        Call address on the stack
```

In New Segment

```
ROUTINE   =       *
          PUSH    A               Save Old segment
          ;-                      Routine code here
          POP     A               Restore old segment
          JMP     TRANSFER        Return address is on top of
                                  stack
```

3) Call Method 2:

In both segments at the same spot

```
LONGCALL  =       *
          PUSH    HL              Save routine address
          LA      DE, LONGLINK
          PUSH    DE              Push Longlink address
          JMP     TRANSFER        GOTO Longlink in the new
                                  segment
LONGLINK  =       *
          POP     HL              Get routine address
          PUSH    A               Save old segment
          LA      DE, LONGRET
          PUSH    DE              Make a psuedo call
          JDR     HL              Jump to routine

LONGRET   =       *
          POP     A               Restore old segment
          JMP     TRANSFER
```

In Old Segment

```
          LI      A,16            New segment
          LA      HL, ROUTINE
          CALL    LONGCALL
```

In New Segment

```
ROUTINE   =       *
          -       Routine code here
          RET     Return to caller (Longret or Local)
```

19.4 SYSTEM POINTER TABLE

The System Pointer Table (SPT) (refer to Table 19-2) is based in the first 16K of memory just as all of the items which are directly pointed to by the SPT. A 16-bit pointer to the starting address of the SPT is located at 3CH (LSB) to 3DH.

Note there are positive offsets and negative offsets for the SPT. To index the offsets you add the offset to the SPT starting address.

Table 19-2. System Pointer Table (Base Address:3CH to 3DH)

	Offset	Size	Type	Name	Description
1)	-338	198	Bytes	SYT.IVEC*	Vector table
2)	-140	1	Byte	SYT.STK*	System stack
3)	-139	7*8	Vector	SYT.IV0-6*	Interrupt Vectors 0-6
4)	-83	2	Address	SYT.IV7*	Interrupt Vector 7
5)	-81	1	Byte	SYT.IQPP*	Propagated System Priority
6)	-80	16*1	Byte	SYT.IQFT*	Interrupt Queue Flag Table
7)	-64	16*2	Address	SYT.ITAB*	Interrupt Table
8)	-32	16*2	Address	SYT.IQUE*	Interrupt Link Table
9)	0	1	Byte	SYP.VERS*	Version
10)	1	1	Byte	SYP.REL*	Release
11)	2	1	Byte	SYP.UPD*	Update
12)	3	1	Byte	SYP.APPL*	Application
13)	4	2	Address	SYP.INTT*	Start-up address
14)	6	1	Byte	SYP.OPT*	Options
15)	7	1	Byte	SYP.STAT*	Status
16)	8	1	Byte	SYP.BSEL*	Boot sub-channel
17)	9	1	Byte	SYP.BCS*	Boot CS
18)	10	1	Byte	SYP.YEAR*	Current year
19)	11	1	Byte	SYP.MONTH*	Current month
20)	12	1	Byte	SYP.DAY*	Current day
21)	13	1	Byte	SYP.HOUR*	Current hour
22)	14	1	Byte	SYP.MIN*	Current minute
23)	15	1	Byte	SYP.SEC*	Current Second

Table 19-2 System Pointer Table (Base Address:3CH to 3DH (Cont.)

	Offset	Size	Type	Name	Description
24)	16	2	Integer	SYP.CCNT*	Clock overflow count
25)	18	2	Address	SYP.CLNK*	Clock queue link
26)	20	2	Integer	SYP.TLIM*	Time slice (milliseconds)
27)	22	2	Integer	SYP.RESL*	Clock resolution (MS)
28)	24	2			Not used
29)	26	2	Address	SYP.CBUF*	Crash dump area
30)	28	1	Byte	SYP.IIC*	Illegal interrupt count
31)	29	1	Byte	SYP.CCODE*	Crash code
32)	30	1	Byte	SYP.IL*	Interrupt Level
33)	31	1	Byte	SYP.CS*	Channel Select
34)	32	1	Byte	SYP.TPRI*	Current task priority
35)	33	1	Byte	SYP.TNR*	Current task number
36)	34	2	Address	SYP.TCB*	Current task TCB
37)	36	2	Address	SYP.STK*	Current system stack
38)	38	2	Address	SYP.SQUE*	Head system RCB queue
39)	40	2	Address	SYP.SQND*	Tail system RCB queue
40)	42	2	Address	SYP.RQUE*	Head ready TCB list
41)	44	2	Address	SYP.ILNK*	Clock interval queue
42)	46	2	Address	SYP.SLNK*	Time (HMS) Queue
43)	48	2	Address	SYP.TLNK*	Time of day queue
44)	50	2	Address	SYP.NODE*	Node pool list
45)	52	2	Address	SYP.FCB*	FCB pool list
46)	54	2	Address	SYP.SRTQ*	SVC call reference table
47)	56	2	Address	SYP.SZTQ*	SVC2 call reference table
48)	58	2	Address	SYP.VMTQ*	Volume mnemonic queue
49)	60	2	Address	SYP.VRTQ*	Volume reference queue
50)	62	2	Address	SYP.DMTQ*	Device mnemonic queue
51)	64	2	Address	SYP.DRTQ*	Device reference queue
52)	66	2	Address	SYP.TMTQ*	Task Mnemonic queue
53)	68	2	Address	SYP.TATQ*	Task reference queue
54)	70	2	Address	SYP.GMTQ*	General mnemonic queue
55)	72	2	Address	SYP.GRTQ*	General reference queue
56)	74	2	Address	SYP.TCBT*	TCB table
57)	76	2	-	-	Reserved
58)	78	2	Address	SYP.VOLN*	Volume Name List

Table 19-2. System Pointer Table (Base Address:3CH to 3DH (Cont.)

	Offset	Size	Type	Name	Description
59)	80	2	Address	SYP.FMGB*	File manager buffers link
60)	82	2	Integer	SYP.BUFX*	Number of free file buffers
61)	84	2	Address	SYP.MEMQ*	Free memory queue
62)	86	2	Address	SYP.UTOP*	Top of memory after SYSINIT
63)	88	2	Address	SYP.MBOT*	Memory base
64)	90	2	Address	SYP.MTOP*	Memory limit
65)	92	2	Address	SYP.TBOT*	System table base
66)	94	2	Address	SYP.TTOP*	System table limit
67)	96	2	Address	SYP.SBOT*	System code base
68)	98	2	Address	SYP.STOP*	System code limit
69)	100	4	String	SYP.SVOL*	System volume name
70)	104	4	String	SYP.SPVL*	Spool volume name (not used)
71)	108	4	String	SYP.RVCL*	Roll volume name (not used)
72)	112	2	-	SYP.CCAL*	-
73)	114	2	Address	SYP.CADR*	Crash PC
74)	116	2	Address	SYP.CSTK*	Crash SP
75)	118	20	Data	SYP.CREG*	Crash registers AF, BC, DE, HL, Y, X AF1, BC1, DE1, HL1
76)	138	1	Byte	SYP.SYS*	System port copy
77)	139	1	Byte	SYP.PBA*	Program MAP A copy
78)	140	1	Byte	SYP.PBB*	Program MAP B copy
79)	141	1	Byte	SYP.HCOL*	Display control copy
	Total	480			

In examining the SPT the following points should be kept in mind:

1) The NIL pointer values is zero. Nil is assigned to a pointer that designates a non-existant value. It is used at the end of a linked list.

2) Assign and request queues in various control blocks are not used.

3) The RCAD count in an FCB indicates the number of tasks using it. The write count is not used.

4) The TCB logical unit queue (.LUQ*) contains the head pointer to the list of FCB's and nodes allocated to a task. The nodes are in turn linked to RCB's of the associated, allocated resource.

5) FCB read/write count values:
   -1 exclusive access
    1 no active access
  1-127 shared access

6) The SVC and SVC2 in the SPT list formats for SPT.SVC* and SPT.SVC2* have either RRT's for re-entrant resources or RCB's and RRT's for non-reentrant, queued resources. There are no associated RMT's.

7) GMT* and GRT* in the SPT are not used.

8) TCB table (in SPT format:
   16-2 byte entries, one for each task (0-15) which print to the TCB for a task. The table is located at 3EF0 (Hex).

9) All non-system interrupt drivers must reside in a programs A-segment.

19.5 RESOURCE ATTRIBUTES WORD

The resource attribute word (refer to Table 19-2) is used in various structures pointed to by the SPT. If a bit is 1 n the word then, the resource can perform the indicated function.

Table 19-3. Resource Attribute Word

Bit	Name	Description
0	ATR.READ*	Read
1	ATR.WRIT*	Write
2	ATR.FASC*	Formatted ASCII
3	ATR.SPEC*	Special Formatting
4	ATR.RND*	Random Access
5	ATR.IACT*	Interactive Device
6		Reserved
7		Reserved
8		Reserved
9	ATR.FR*	Forward record
10	ATR.FF*	Forward File
11	ATR.WF*	Write File Mark
12	ATR.BR*	Backward Record
13	ATR.BF*	Backward File
14	ATR.RW*	Rewind
15	ATR.ATTN*	Attention
16		Reserved

19.6 RESOURCE MNEMONIC TABLE (RMT) AND RESOURCE REFERENCE TABLE (RRT

The Resource Mnemonic Table (RMT) (refer to Table 19-4) lists symbolic definitions of the resources that are to be used. It is used in the DMT, TMT, VMT, and GMT. The REsource Reference Table (RRT) (refer to Table 19-5) gives numeric definitions of a resource; in particular, it contains the number and type of the resource. It is used in the DCT, TRY, VET, and GRT. It also holds the assign linkage and entry to the handler on shared resources, or the address of the Resource Control Block (RCB) or exclusive resources. The name of the relevant resource and address of the RRT is contained in the RMT.

Table 19-4. Resource Mnemonic Table

	Offset	Size	Type	Name	Description
1)	0	2	Address	RMT.MQLK*	Next RMT in list
2)	1	4	String	RMT.NAME*	Name (4 char)
3)	6	2	Address	RMT.RRT*	Associated RRT
	Total	8			

Table 19-5. Resource Reference Table

	Offset	Size	Type	Name	Description
1)	0	2	Address	RRT.RQLK*	Next RRT in list
2)	2	1	Byte	RRT.RNR*	Resource number
3)	3	1	Byte	RRT.TYPE*	Type (see below)
4)	4	1	Byte	-	Not used
5)	5	1	Byte	RRT.SEG*	RCB segment
6)	6	2	Address	RRT.ADR*	RCB address
7)	8	2	-	-	Not used
8)	10	1	Byte	RRT.RCNT*	Read count
9)	11	1	Byte	RRT.WCNT*	Write count
10)	12	2	Address	RRT.RMT	RMT address
11)	14	2	Address	RRT.AQUE	Assign queue
12)	16	2	Address	RRT.RQUE	Request queue
	Total	18			

3) RRT.TYPE, Type

This byte specifies the type of resource as follows:

Bits	Size	Name	Description
0-1	2	RRT.TYPE*	Type
		RRT.PURE*	Pure resource
		RRT.RCB*	Impure resource (.ADDR)
		RRT.RRT*	Dummy resource (.ADDR)
		RRT.AREA*	Area (.ADDR)
2	1	RRT.DIR*	Directory device
3	1	RRT.SVC*	Enter on SVC call
4	1	RRT.OFFL*	Offline
5	1	RRT.PROT*	Protected resource
6	1	RRT.NFST*	Non-file structured
7	1	RRT.NAB*	Non-abortable

19.7 BUFFER CONTROL NODE

Buffers are useful for free space allocation. The table for the buffer control node has the following structure:

	Offset	Bytes	Type	Name	Description
1)	0	2	Address	BUF.BLNK*	Buffer link (in list)
2)	2	1	Byte	BUF.FLAG*	Buffer flag
3)	3	1	Byte	BUF.UCNT*	Buffer Usage Count or
4)	4	4	Long Integer	BUF.DADR*	Associated device or sect

The byte allocation for the Buffer Flag is as follows:

	Bit	Name	Description
1)	0	BFL.FREE*	Buffer is free
2)	1-4	-	Reserved
3)	5	BFL.INTR*	Buffer in transfer to disk
4)	6	BFL.UPD*	Buffer is updated & must be saved
5)	7	BFL.LRU*	Buffer is the last used

19.8 RESOURCE CONTROL BLOCK (RCB)

This RCB holds all of the information pertaining to an exclusive resource such as the status of the resource, the request queue into the resource and entry into the resource. The RCB is normally at the head of all control blocks in the operating system. The Resource Control Block Table (refer to Table 19-6) is used in the DCB, TCB, FCB, and VCB.

Table 19-6. Resource Control Block Table

	Offset	Size	Type	Name	Description
1)	-20	2	Address	RCB.SQLK*	System queue link
2)	-18	2	Address	RCB.CREQ*	Current request
3)	-16	1	Byte	RCB.CPRI*	Current priority
4)	-15	1	Byte	RCB.CTCB*	Current TCB number
5)	-14	1	Byte	RCB.TYPE*	Resource type (See below)
6)	-13	1	Byte	RCB.STATE*	Resource status (See below)
7)	-12	2	Address	RCB.RQUE*	Request queue
8)	-10	2	Address	RCB.RRT*	Numeric reference
9)	-8	2	Address	RCB.PRNT*	Coordination address
10)	-6	1	Byte	-	
11)	-5	1	Byte	RCB.SEG*	Resource segment
12)	-4	2	Address	RCB.INIT*	Resource initiator address
13)	-2	2	Address	RCB.TERM*	Resource terminator address
	Bytes	20			

5) RCB.TYPE, Type

The byte allocation for the resource type is as follows:

Bit		Name	Description
0-2		RCT.TYPE*	'RCB' identifier
	0	RCT.RCB*	
	1	RCT.DCB*	
	2	RCT.TCB*	
	3	RCT.FCB*	
	4	RCT.VCB*	
	5	RCT.AREA*	
3		RCT.PRNT*	Parent present
4		RCT.DESC*	Descendent present
5		RCT.NEW*	Don't support no wait
6		RCT.PRO*	Don't support protect
7		RCT.NAB*	Don't support abort

6) RCB.STAT, Status

The byte alloation for the Resource status is as follows:

Bit	Name	Description
0	RCS.BUSY*	Resource busy
1	RCS.OFFL*	Resource offline
2	RCS.CAN*	Resource cancelled
3	RCS.NW*	Resource no wait in program
4	RCS.ONSQ*	present on system queue

19.9 DEVICE CONTROL BLOCK (DCB)

The DCB is used by the Connection, Disconnection, and System Interrupt Handlers to specify the characteristics of each device in the operating system and to serve as work space for device drivers during an I/O request.

The DCB Table consists of the minus offsets in the RMT, RRT, CCB, ICB, RCB, DMT, and DRT (refer to Table 19-7).

Table 19-7. Device Control Block Table

		Offset	Size	Type	Name	Description
RMT	1)	-42	2	Address	DMT.MQLK*	Next DCBPMT
	2)	-40	4	String	DMT.NAME*	Name
	3)	-36	2	Address	DMT.RRT*	RRT Link
RRT	4)	-34	2	Address	DRT.RQLK*	DRT Link
	5)	-32	1	Byte	DRT.RNR	Resource Number
	6)	-31	1	Byte	DRT.TYPE	Resource Type
	7)	-30	2	Address	DRT.ADR	Entry RCB Address
	8)	-28	1	Byte	DRT.RCNT	Read Count
	9)	-27	1	Byte	DRT.WCNT	Write Count
	10)	-26	2	Address	DRT.RMT	RMT Address
	11)	-24	2	Address	DRT.AQUE	Assign Queue
	12)	-22	2	Address	DRT.RQUE	Request Queue
CCB	13)	-20	12	-	-	CCB
ICB	14)	8	8	-	-	ICB
RDB	15)	0	2	Word	DCB.ATTR*	Attributes
	16)	2	2	Integer	DCB.RECL*	Rec. Length
	17)	4	1	Byte	DCB.CODE*	Resource code
	18)	5	1	Byte	DCB.SNR*	Sys. Num. d/task#)
	19)	6	1	Byte	DCB.TYPE*	Res. type (see belowow)
	20)	7	1	Byte	DCB.STAT*	Res. status (see bel
	21)	8	2	Address	DCB.FMTE*	Data format routine
	22)	10	1	Byte	DCB.QPAR*	Number of parameters
	23)	11	1	Byte	DCB.QSEG*	SVC B seg.
	24)	12	2	Address	DCB.QSVC*	SVC param. block add
	25)	14	1	Byte	DCB.QFC*	SVC function
	26)	15	1	Byte	DCB.QLU*	SVC status
	27)	16	1	Byte	DCB.QTS*	SVC LU
	28)	17	1	Byte	DCB.QTS*	Req. term. status
	29)	18	2	Address	DCB.QBAD*	Req. buffer address
	30)	20	2	Integer	DCB.QBSZ*	Req. buffer size
	31)	22	2	Integer	DCB.QBCN*	Req. byte count
	32)	24	4	Long Int.	DCB.QRND*	Req. Rec. Address
	33)	26	-	-	DCB.OPT*	Device, Option Data
		Total	68			

19) DCB.TYPE, Device Type: The byte allocation for the Device Type is as follows:

Bit	Name	Description
0	DCT.TCB*	TCB present
1	DCT.DEDI*	Dedicated service
2	DCT.ENI*	Runs with interrupts enabled
3	DCT.NOIG*	No interrupt link
4	DCT.TASK*	Task device
5	DCT.DUAL*	Dual DCB?

20) DCB.STAT, Device Status: The byte allocation for the Device status is as follows:

Bit	Size	Name	Description
0	1	DCS.INT*	Init. on device
1	1	DCS.TIME*	Treat T.ONT as interrupt
2	1	DCS.TOUT*	Resource has gone to Timeout

19.10 INTERRUPT CONTROL BLOCK (ICB)

This table (See Table 19-8) is used by the interrupt system and by the real time handler. It is an addition to the RCB.

Table 19-8. ICB Table

	Offset	Size	Type	Name	Description
1)	-28	2	Address	ICB.IQLK*	Next ICB
2)	-26	1	Byte	ICB.PRIO*	ICB priority
3)	-25	1	Byte	ICB.IL*	Interrupt level
4)	-24	1	Byte	ICB.TYPE*	Type (below)
5)	-23	1	Byte	ICB.STAT*	Status (below)
6)	-22	2	Address	ICB.CON*	Continuator address
	Total	8			

4) ICB.TYPE, Type: This byte specifies the ICB as follows:

Bit	Name	Description
0	ICT.CCB*	"CCB" present
1	ICT.NOIQ*	No interrupt queue

5) ICB.STAT, Status: This byte specifies the current status as follows:

Bit	Name	Description
0	ICS.ONIQ*	Present on Int. queue
1	ICS.SINT*	Software int. generated
2	ICS.TOUT*	Time-out generated

19.11 CHANNEL CONTROL BLOCK (CCB)

The Channel Control Block is used by the interrupt system to scan the interfaces (refer to Table 19-9).

Table 19-9. Channel Control Block Table

	Offset	Size	Type	Name	Description
1)	-40	1	Byte	CCB.CS*	Device channel Add.)
2)	-39	1	(not used)	CCB.TM*	Interrupt Mask
3)	-38	1	Byte	CCB.FLG*	Flags
4)	-37	1	Byte	CCB.XOR*	XOR Mask
5)	-36	2	Integer	CCB.TLIM*	Time-out time limit
6)	-34	2	Integer	CCB.TCNT	Time-out counter
7)	-32	2	Integer	CCB.TMND	Time-out handler
8)	-30	2	Address	CCB.SUBC	Continuation handler
	Total	12			

19.12 TASK CONTROL BLOCK (TCB)

The TCB is used to identify the name, location and type of tasks. The structure of the TCB is shown in Table 19-10.

Table 19-10. Task Control Block Table

	Offset	Byte	Type	Name	Description
1)	-40	2	Address	TMT.MQLK	Mnemonic linked list
2)	-38	4	String	TMT.NAME	Task name
3)	-34	2	Address	TMT.RRT	Pointer to RRT
4)	-32	2	Address	TRT.RQLK	Next task link
5)	-30	1	Byte	TRT.RNR	Resource number
6)	-29	3	-	-	Not used
7)	-26	2	Address	TRT.ADDR	RCB address
8)	-24	1	Byte	TRT.RCNT	Read assign count
9)	-23	1	Byte	TRT.WCNT	Write assign count
10)	-22	2	Address	TRT.RMT	RMT pointer
11)	-20	2	Address	TRT.AQUE	Assign queue
12)	-18	2	Address	TRT.RQUE	Request queue
13)	-16	16	Block	-	RCB
14)	0	1	Byte	TCB.TYPE*	Task type
15)	1	1	Byte	TCB.STAT*	Task status
16)	2	1	Byte	TCB.OPT*	Task options
17)	3	1	Byte	TCB.PEND*	Task pending status
18)	4	1	Byte	TCB.TPRT*	Task priority
19)	5	1	Byte	TCB.RCOD*	Task return code
20)	6	1	Byte	TCB.MODE*	Task mode
21)	7	1	Byte	TCB.KNLV*	Kernal level cntr.
22)	8	1	Byte	TCB.EVC*	Task event cntr.
23)	9	1	Byte	TCB.TCOD*	Traced code
24)	10	2	Address	TCB.TADR*	Traced address
25)	12	2	Address	TCB.PURE*	Ptr. TCB of pure code

Table 19-10. Task Control Block Table (Cont.)

	Offset	Byte	Type	Name	Description
26)	14	2	Address	TCB.MBOT*	Low memory limit
27)	16	2	Address	TCB.UBOT*	First free byte allocated
28)	18	2	Address	TCB.UTOP*	Top of code
29)	20	2	Address	TCB.SBOT*	Bottom of stack
30)	22	2	Address	TCB.MTOP*	High memory limit
31)	24	2	Address	TCB.MEMQ*	Allocated memory pod
32)	26	1	Byte	TCB.ASEG*	Task A seq. loc.
33)	27	1	Byte	TCB.BSEG*	Task B seq. loc.
34)	28	1	Byte	TCB.MLIM*	Task Mode & limit
35)	29	1	Byte	TCB.ASIZ*	Task A seq Size (4K)
36)	30	1	Byte	TCB.BSIZ*	Task B seq size
37)	31	1	Byte	TCB.PTSK*	Parent task #
38)	32	2	Address	TCB.SADR*	Task start address
39)	34	2	Address	TCB.RQLK*	Ready queue link
40)	36	1	Byte	TCB.DPRI*	Dispatch priority
41)	37	1	Byte	TCB.TNR*	Task number
42)	38	2	Address	TCB.STK*	Current stack
43)	40	2	Address	TCB.TRMQ*	Task wait a
44)	42	2	Address	TCB.STSQ*	Task status change Q
45)	44	2	Address	TCB.WQUE*	Task wait a
46)	46	2	Address	TCB.LUQ*	Task LU Q
47)	48	2	Integer	TCB.TLIM*	Time slice limit
48)	50	2	Integer	TCB.TCNT*	Time slice count
49)	52	2	Address	TCB.UDA*	User dedicated area
50)	54	1	Byte	TCB.NNOD*	Number of reg. nodes
51)	56	1	Byte	TCB.UNOD*	Number of nodes left
52)	57	1	Byte	TCB.NFCB*	Number of FCB's
53)	58	1	Byte	TCB.UFCB*	Number of FCB's left
	Total	98			

14) TCB.TYPE, Type: The byte allocation for type is:

Bit	Name	Description
0	TCT.RES*	Resident
1	TCT.NAB*	Non-abortable
2	TCT.ROLL*	Rollable
3	TCT.PURE*	Pure code
4	TCT.ETSK*	E-task
5	TCT.NTRC*	Not tracable

15) TCB.STATUS, Status: The byte allocation for status is:

Bit	Name	Description
0	TCS.ACT*	Task active
1	TCS.ONRQ*	Task on ready queue
2	TCS.WAIT*	Task waiting
3	TCS.WDON*	Task waiting for complete
4	TCS.PAUS*	Task paused
5	TCS.CAN*	Task cancelled
6	TCS.TOUT*	Task time sliced elapsed

16) TCB.OPTION, Option: The byte allocation for option is:

Bit	Name	Description
0	TCO.DASG*	Default assign allowed
1	TCO.NSTK*	No stack check
2	TCO.EMSG*	System error message
3	TCO.RCOV*	System recovery
4	TCO.DMP*	Dump on abort

17) TCB.MODE, Mode: The byte allocation for mode is:

Bit	Name	Description
0	TCM.FMGR*	In file manager
1	TCM.PEND*	In pending status
2	TCM.WAIT*	In connect/disconnect
3	TCM.STKV*	Stack violation
4	TCM.TRAC*	Trace on

19.13 RESOURCE DESCRIPTOR TABLE (RDT)

Table 19-11 shows the contents of the RDT Table.

Table 19-11. RDT Table

	Offset	Bytes	Type	Name	Description
1)	1-16	16	Block		RCB
2)	0	16	Block	RDT.RRT*	RRT
3)	16	1	Byte	RDT.STAT*	Status
4)	17	1	Byte	RDT.FLGS*	Flags
5)	18	4	String	RDT.VOLN*	Volume name
6)	22	12	String	RDT.NAME*	File/directory name
7)	34	8	String	RDT.ELMT*	Element name
	Total	58			

3) ROT.STATUS, Status: This byte of the RDT contains the following:

Bit	Name	Description
0	RDS.PRES*	Present
1	RDS.INTR*	In transfer
2	-	Reserved

19.14 FILE CONTROL BLOCK (FCB)

The FCB is shown in Table 19-12.

Table 19-12. File Control Block Table

		Offset	Bytes	Type	Name	Description
(RMT)	1)	-40	2	Address	FMT.MQLK*	Next mnemonic link
	2)	-38	4	String	FMT.NAME*	Name
	3)	-34	2	Address	FMT.RRT*	Link to
(RRT)	4)	-32	2	Address	FRT.RQLK*	RRT list link
	5)	-30	1	Byte	FRT.RNR*	Resource number
	6)	-29	2	Word	FRT.TYPE*	Type
	7)	-27	1	Byte	FRT.SEQ*	Entry segment Add.
	8)	-26	2	Address	FRT.ADR*	RCB/Entry Address
	9)	-24	1	Byte	FRT.RCNT*	Read count
	10)	-23	1	Byte	FRT.WCNT*	Write Count
	11)	-22	2	Address	FRT.RMT*	RMT address
	12)	-20	2	Address	FRT.AQUE*	Assign queue
	13)	-18	2	Address	FRT.RQUE*	Request queue
(RCB)	14)	-16	16	Segment	FCB.RCB*	Connection block
(FCB)	15)	0	2	Word	FCB.ATTR*	Attributes
	16)	2	2	Integer	FCB.RECL*	Record length (pcvar)
	17)	4	1	Byte	FCB.CODE*	Code
	18)	5	1	Byte	FCB.SDNR*	System device number
	19)	6	1	Byte	FCB.MOD*	Modifier
	20)	7	1	Byte	FCB.FSTA*	Status
	21)	8	2	Word	-	Reserved
	22)	10	1	Byte	FCB.QPAR*	Number of parameters
	23)	11	1	Byte	FCB.QSEG*	SVC seg. add.
	24)	12	2	Address	FCB.QSVC*	SVC. param. add.
	25)	14	1	Byte	FCB.QFC*	SVC func. code
	26)	15	1	Byte	FCB.QRS*	SVC return status
	27)	16	1	Byte	FCB.QLU*	SVC logical unit
	28)	17	1	Byte	FCB.QMOD*	SVC modifier

Table 19-12. File Control Block Table (Cont.)

	Offset	Bytes	Type	Name	Description
29)	18	2	Address	FCB.QBAD*	SVC Buffer Address
30)	20	2	Integer	FCB.QBCN*	SVC byte count
31)	22	2	Integer	FCB.QRND*	SVC random record address
32)	24	4	Integer	FCB.QSIZ*	SVC file size
33)	28	2	Address	FCB.VCBP*	Volume name (RMT)
34)	30	1	Byte	FCB.RCM*	Read count mode
35)	31	1	Byte	FCB.WCM*	Write count mode
36)	32	4	Integer	FCB.MDIR*	Master File Dir Index
37)	36	4	Integer	FCB.DTRA*	Own file directory index
38)	40	4	Integer	FCB.MODE*	File mode
39)	44	4	Integer	FCB.CXAD*	Current index sector address
40)	48	4	Integer	FCB.LXAD*	Last index sector address
41)	52	2	Integer	FCB.LSEG*	Seq # of last index sec
42)	54	14	Integer	FCB.EOFP*	EOF random address
43)	58	4	Integer	FCB.WRND*	Work random addresss
44)	62	2	Word	FCB.FLAG*	File flag
45)	64	4	Integer	FCB.CDAD*	Current data sector address
46)	68	2	Integer	FCB.XSEG*	Sequence number if current sector
47)	70	2	Address	FCB.BUFR*	Pointer to data buffer mode
48)	72	4	Integer	FCB.BASE*	Base of current sector
49)	76	4	Integer	FCB.HRND*	Highest random address
50)	80	2	Integer	FCB.CLUZ*	File cluster size
51)	82	2	Integer	FCB.BLK*	File block size (in clusters)
	Total	124			

20) FCB.STATUS, FCB Status (Byte): This byte contains the following:

Bit	Value	Name	Description
0		FST.DBIU*	Data buffer in use
1		FST.IBIU*	Index buffer in use
2		FST.TEMP*	Temp. file (delete at close)
3		FST.NSA*	New space allocated
4		FST.EOF*	E-O-file
5		FBM.UPD*	Data buffer updated
6-7		FBM.MASK*	FBM mask
	0	FBM.SYSB*	System buffering
	1	FBM.USER*	User buffering
	2	FBM.NOB*	No buffering
	3	FBM.BYTE*	Byte access

44) FCB.FLAG*, File Flag (Word): This byte contains the following:

Bit	Name	Description
0	FFL.POSN*	Positioning req'd before index
1	FFL.ELMT*	Doing element flag
2	FFL.EXCL*	In exclusive mode
8	FFL.CONT*	Contiguous file
9	FFL.INDX*	Indexed file
10	FFL.CYC	Cyclical file

19.15 VOLUME DESCRIPTOR SECTOR (VDS)

The VDS describes the first sector of a disk. Table 19-13, shows its contents.

Table 19-13. VDS Table

	Offset	Bytes	Type	Name	Description
1)	0	4	String	VDS.VOLN*	Volume name
2)	4	1	Byte	-	Reserved
3)	5	3	Med. Int.	VDS.MFOP*	Master dir. file index
4)	8	4	-	-	Reserved
5)	12	1	Byte	VDS.ALOL*	Length of allocation tab
6)	13	3	Med. Int.	VDS.ALOP*	Allocation table index
7)	16	2	Integer	VDS.CLUZ*	Def. cluster size (in shift cnt)
8)	18	2	Integer	VDS.BLKB*	Def. block size in clusters
19)	20	2	Integer	VDS.MAXL*	Def. maximum length in sect.
10)	22	1	Byte	VDS.FLAG*	Volume flag (low)
11)	23	1	Byte		Volume Flag (high)
12)	24	8	V. Lg. Int.	VDS.OSPT*	OS File index
13)	32	4	Long Int.	VDS.MXSA*	Maximum sector add.
14)	36	2	Integer	VDS.MXSC*	Sectors/cylinder
5)	38	2	Integer	VDS.MXTC*	Tracks/cylinder
		40	Bytes		

10) VDS.FLAG*, Volume Flag (Low): This byte contains:

Bit	Name	Description
0	VSF.OSPR*	Operating System Present

11) Volume Flag (High): This byte contains:

Bit	Name	Description
0	VSF.NOFP*	Volume safe at mark on flag (delete marks ignored)

19.16 VOLUME CONTROL BLOCK (VCB)

The VCB (refer to Table 19-14) changes all names that being with F (i.e. FCB & FRT) to V (i.e. VCB & VRT). It is identical to the FCB.

Table 19-14. File Control Block Table

		Offset	Bytes	Type	Name	Description
(RMT)	1)	-40	2	Address	FMT.MQLK*	Next mnemonic link
	2)	-38	4	String	FMT.NAME*	Name
	3)	-34	2	Address	FMT.RRT*	Link to
(RRT)	4)	-32	2	Address	FRT.RQLK*	RRT list link
	5)	-30	1	Byte	FRT.RNR*	Resource number
	6)	-29	2	Word	FRT.TYPE*	Type
	7)	-27	1	Byte	FRT.SEQ*	Entry segment Add.
	8)	-26	2	Address	FRT.ADR*	RCB/Entry Address
	9)	-24	1	Byte	FRT.RCNT*	Read count
	10)	-23	1	Byte	FRT.WCNT*	Write Count
	11)	-22	2	Address	FRT.RMT*	RMT address
	12)	-20	2	Address	FRT.AQUE*	Assign queue
	13)	-18	2	Address	FRT.RQUE*	Request queue
(RCB)	14)	-16	16	Segment	FCB.RCB*	Connection block
(FCB)	15)	0	2	Word	FCB.ATTR*	Attributes
	16)	2	2	Integer	FCB.RECL*	Record length (pcvar)
	17)	4	1	Byte	FCB.CODE*	Code
	18)	5	1	Byte	FCB.SDNR*	System device number
	19)	6	1	Byte	FCB.MOD*	Modifier
	20)	7	1	Byte	FCB.FSTA*	Status
	21)	8	2	Word	-	Reserved
	22)	10	1	Byte	FCB.QPAR*	Number of parameters
	23)	11	1	Byte	FCB.QSEG*	SVC seg. add.
	24)	12	2	Address	FCB.QSVC*	SVC. param. add.
	25)	14	1	Byte	FCB.QFC*	SVC func. code
	26)	15	1	Byte	FCB.QRS*	SVC return status
	27)	16	1	Byte	FCB.QLU*	SVC logical unit
	28)	17	1	Byte	FCB.QMOD*	SVC modifier

Table 19-14. File Control Block Table (Cont.)

	Offset	Bytes	Type	Name	Description
29)	18	2	Address	FCB.QBAD*	SVC Buffer Address
30)	20	2	Integer	FCB.QBCN*	SVC byte count
31)	22	2	Integer	FCB.QRND*	SVC random record address
32)	24	4	Integer	FCB.QSIZ*	SVC file size
33)	28	2	Address	FCB.VCP*	Volume name (RMT)
34)	30	1	Byte	FCB.RCM*	Read count mode
35)	31	1	Byte	FCB.WCM*	Write count mode
36)	32	4	Integer	FCB.MDIR*	Master File Dir Index
37)	36	4	Integer	FCB.DTRA*	Own file directory index
38)	40	4	Integer	FCB.MODE*	File mode
39)	44	4	Integer	FCB.CXAD*	Current index sector address
40)	48	4	Integer	FCB.LXAD*	Last index sector address
41)	52	2	Integer	FCB.LSEG*	Seq # of last index sec
42)	54	14	Integer	FCB.EOFP*	EOF random address
43)	58	4	Integer	FCB.WRND*	Work random addresss
44)	62	2	Word	FCB.FLAG*	File flag
45)	64	4	Integer	FCB.CDAD*	Current data sector address
46)	68	2	Integer	FCB.XSEG*	Sequence number if current sector
47)	70	2	Address	FCB.BUFR*	Pointer data buffer mode
48)	72	4	Integer	FCB.BASE*	Base of current sector
49)	76	4	Integer	FCB.HRND*	Highest random address
50)	80	2	Integer	FCB.CLUZ*	File cluster size
51)	82	2	Integer	FCB.BLK*	File block size (in clusters)
	Total	124			

20) VCB Flags: This byte contains the following:

Bit	Name	Description
0	VCF.ABIU*	Allocation buffer in use
1	VCF.AUPD*	Allocation buffer updated
2	VCF.ALLO*	Allocation in process
3-7	-	Reserved

SECTION 20

FILE STRUCTURES

SECTION 20
FILE STRUCTURES

20.1 INTRODUCTION

This section describes the logical layout of the disk, the structure of the Master File Directory, and the structure of ISAM files.

20.2 LOGICAL LAYOUT OF DISK

The logical layout of the disk is shown in Figure 20-1.

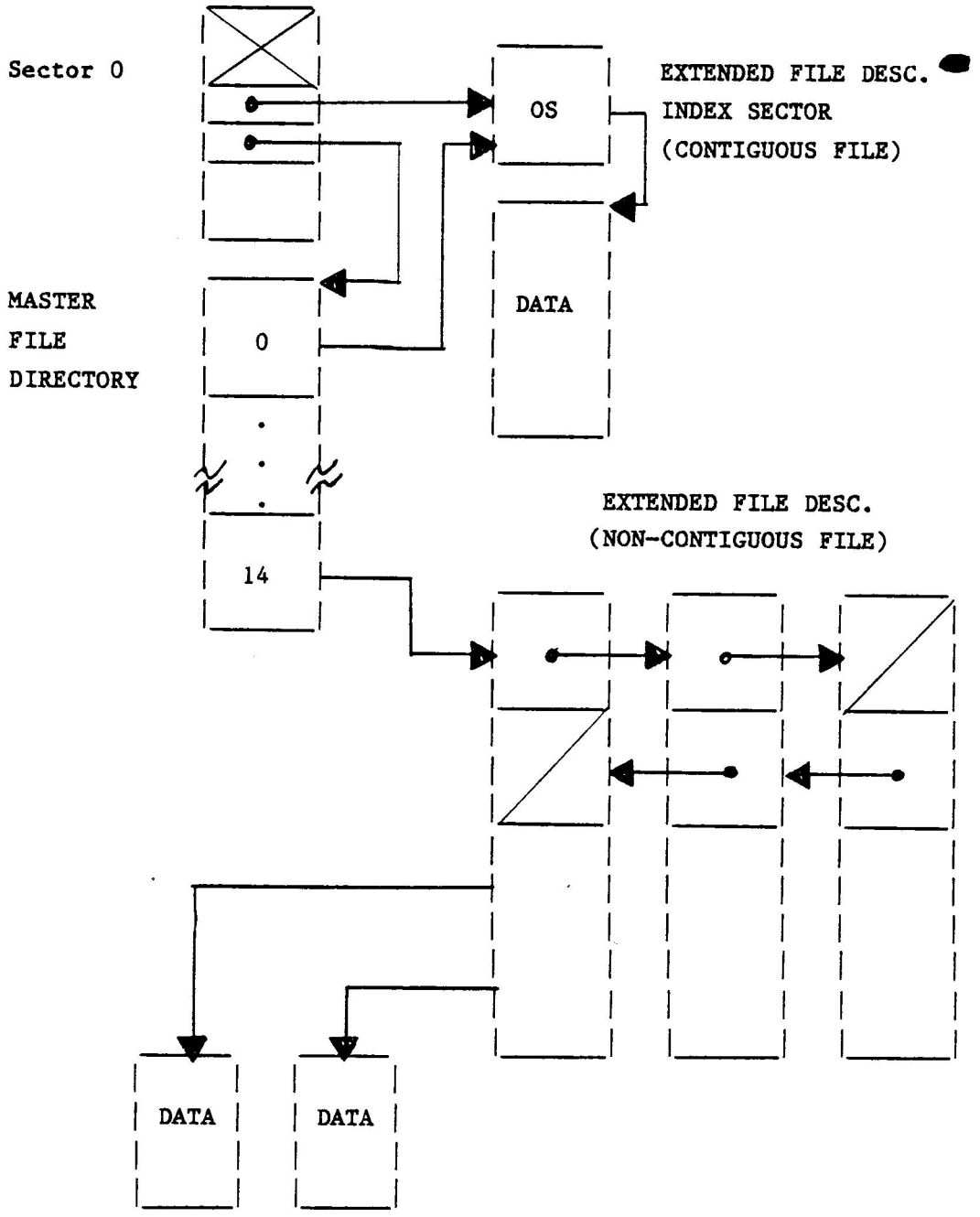


Figure 20-1. Logical Disk Layout

Directory Structure

	Offset	Bytes	Type	Name	Description
1)	0	1	Byte	-	Must be 0 for directory
2)	1	1	Byte	DIRH.CNT*	Entires in directory
3)	2	2	Word	-	Reserved
4)	4	4	Long Int.	DIRS.HPT*	Hash Sector Index
5)	8	8	-	-	Reserved
6)	16	20	Block	-	Directory entry - Directory entry 1-10
7)	236	20	Block	-	Directory entry 1-11
		256 Bytes			

12 Directory entries/sector
15 Sectors/directory
180 Entries/directory

Directory Entry

	Offset	Bytes	Type	Name	Description
1)	0	2	Word	GDE.VLNK	Version linkage
2)	2	1	Byte	GDE.RCNT	Read count
3)	3	1	Byte	GDE.WCNT	Write count
4)	4	12	String	GDE.NAME	Name
5)	16	4	Long Int.	GDE.FXAD	First index sector
		20 Bytes			

Extended File Descriptor (XFD) First Index Sector

	Offset	Bytes	Type	Name	Description
1)	0	4	Long Int.	-	Forward link (next index)
2)	4	4	Long Int.	-	Backward link, must be 0
3)	8	8	Block	-	Reserved
4)	16	4	Long Int.	XFD.LXAD*	Last Index Sector
5)	20	2	Integer	XFD.CSEG*	
6)	22	4	Long Int.	XFD.EOF*	
7)	26	4	Long Int.	XFD.NREC*	
8)	30	2	Word	XFD.FLAG*	
9)	32	2	Integer	XFD.RECL*	Record length (0=var)
10)	34	4	Long Int.	XFD.CHKP*	Check point
11)	38	6	String	XFD.CRDT*	Creation date & time
12)	44	6	String	XFD.LADT*	last assign date & time
13)	50	6	String	XFD.LUDT*	Last update date & time
14)	56	4	Long Int.	-	Data record 0 sec index
15)	-	192	Long Int.	-	
16)	252	4	Long Int.	-	Data record 49 sector
		256 Bytes			

Subsequent Index Sectors

	Offset	Size	Type	Name	Description
1)	0	4	Long Int.	-	Forward link (last one = 0)
2)	4	4	Long Int.	-	Backward link
3)	8	4	Long Int.	-	Data record on sector index
4)	12	240	Long Int.	-	
5)	252	4	Long Int.	-	Data record N+61 sector index
		256 Bytes			

8) XFD.FLAG, File Flag

Bit	Name	Description
0	XFF.CONT*	1 = Contiguous File
		0 = Non-contiguous File

20.3 ISAM FILE STRUCTURE

ISAM, Indexed Sequential Access Method, is a technique used for indexed access to large data files. It can be used for random access using a key string as the search argument, or sequential access using the index.

The data is divided into RECORDS. The records have a fixed, user defineable length, and they are stored in a fixed record length file, the DATA-file. Each data-file has an ISAM-file associated with it.

The ISAM-file may contain up to ten indices into the data file. Each index has a symbolic name. It contains one KEY for each data record. The key consists of a key string, which also is a part of a data record, and a pointer to that record. The keys are ordered within the index to form a B-tree structure.

All record pointers are logical while file reference is symbolic. This means that the data and ISAM files may be copied and utilized on any random access device supported by the operating system.

The ISAM file is initialized by a utility program. After initialization, the ISAM and data files are built by the user using ISAM write operations. Since the index trees are built in a well structured way, there is no need for time consuming reorganizations once the indices are established. The access times will always be at an optimum.

Key Formats

Five different formats are defined for the key strings. The formats are:

1) Binary: This is a string of bytes of selectable length. The string is interpreted as an unsigned binary integer, with most significant byte first.

2) ASCII: This is a string of bytes of selectable length. The bytes are interpreted as 7-bit ASCII characters. Upper and lower case characters have the same value.

3) Integer: This is a string of two bytes, representing a signed integer, the least significant byte first. It is compatible with Monroe BASIC and PASCAL formats.

4) Floating Point: This is a string of four bytes, representing a single precision floating point number. It is compatible with BASIC and PASCAL formats.

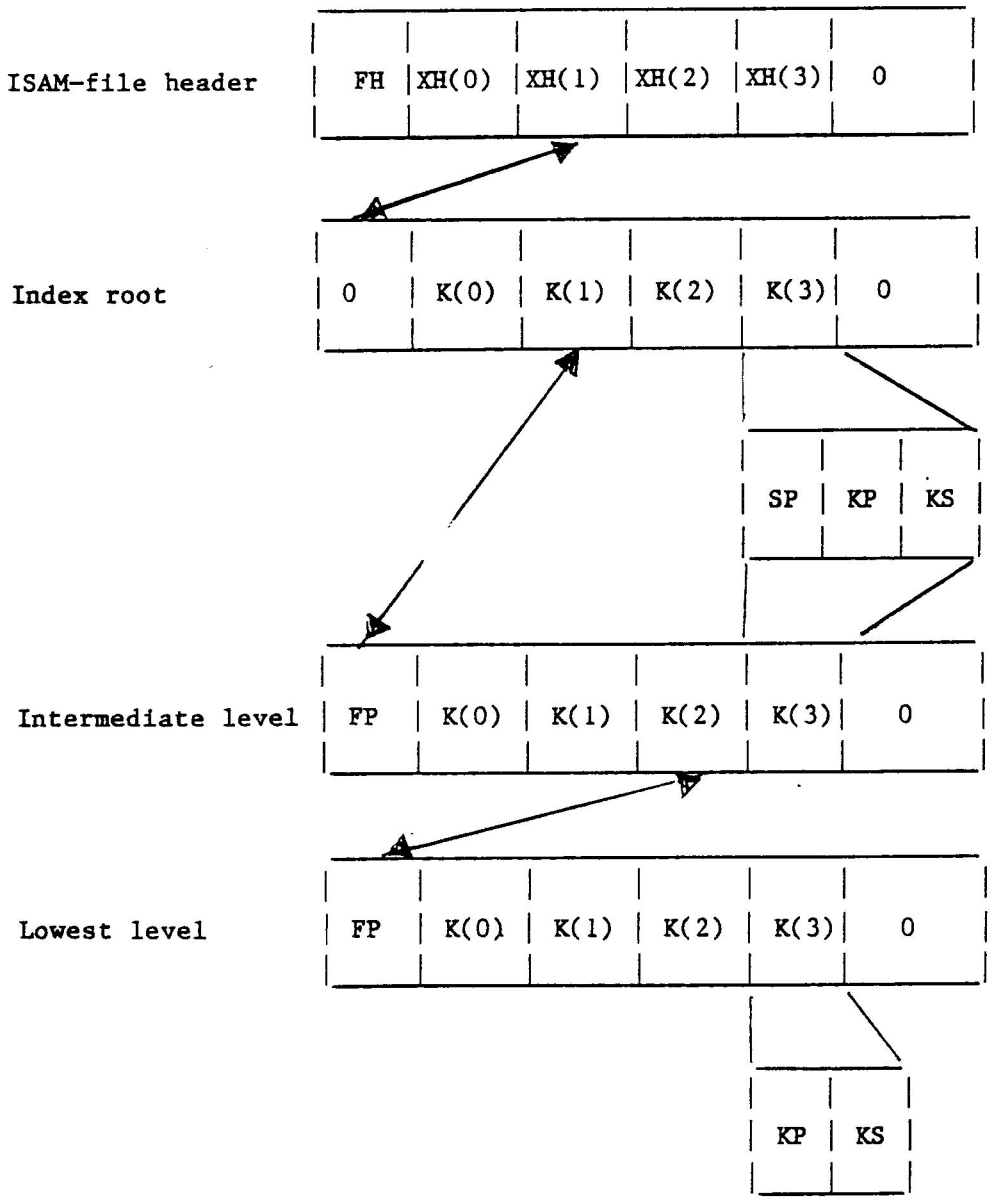
5) Double Precision Floating Point: As above, but string length is eight bytes. it is compatible with the BASIC format.

ISAM-File Format

The ISAM file format is built on the B-tree concept. This concept makes it possible to maintain the search path through the tree at an optimum through insertions and deletions of key items.

The first record of an ISAM-file is a header record (see Figure 20-2). It contains information about the ISAM-file and the data-file it indexes.

An ISAM-file may contain up to ten separate indices with symbolic names. All information about the indices e.g. symbolic name, key type, key position, key length and the B-tree root pointer is stored in the ISAM-file header. The ISAM-file contains one B-tree for each index.



FH = File header
XH = Index header
K = Key
SP = Son pointer
KP = Key pointer (points out the data record)
KS = Key string

Figure 20-2. ISAM File Structure

ISAM File Header Format

The File Header format consisting of a FIle Header and Index descriptor is given as follows:

File Header

The byte allocation for the File Header is as follows:

Byte	Item
0	Version number (-1)
1-29	Data file descriptor
30-33	Root pointer of deleted records chain (index file)
34-37	Root pointer of deleted records chain (data file)
38	First index descriptor

Index Descriptor

The byte allocation for the index descriptor is as follows:

Byte	Item
0-11	Index name
12-15	Index root pointer
16	Index flags (bit 0 : duplicates allowed)
17-18	Key string start post (LSB, MSB)
19	Key string length
20	Key type (0-4. Bit 7: Descending sequence)

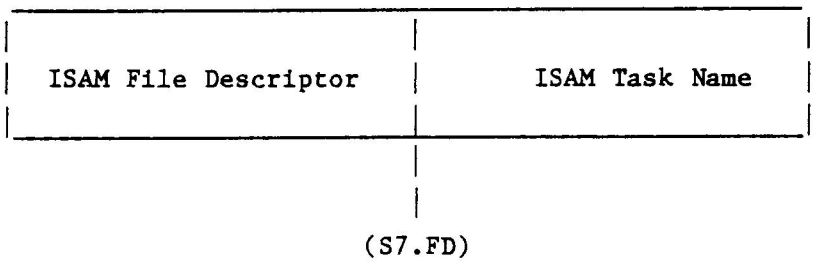
Multi-Task ISAM

The Multi-Task ISAM facility makes it possible for several users to use the same data base. Each user may also use more than one data base. The facility is added to the multi-task operating system by loading and starting an ISAM-task. Each user may then assign to the ISAM task, and the ISAM-task will assign the selected ISAM/data files. All input/output operations will then take place through the ISAM-task which will co-ordinate the different users and their requests.

Assembly Language Interface to ISAM: A user program written in assembly language will interface to the ISAM-task using SVC 7 and SVC 1 calls for assign and I/O functions.

Assign Function: The SVC 7 assign function will assign a user to the ISAM task. The ISAM task will assign the ISAM and data files requested, and will maintain information about the files and the users internally.

The SVC 7 block is a standard parameter block (See section 14 reference manual). The file descriptor pointer however, points at a special file descriptor which defines both the ISAM task (bytes 0-27) and a ISAM file (bytes 28-1).



The file modifier byte (S7.MOD) is used in a special way. If it is set to 255 (decimal) when as assign function is performed, the ISAM task will terminate when a subsequent close function has been performed.

I/O Functions: The I/O functions are performed using SVC 1 calls where the SVC 1 block is extended as follows:

S0.FC	S0.RS	S1.LU	S1.TS
Function Code	Return Status	Logical Unit	Term Status
S1.BAD		S1.BSZ	
Buffer address		Buffer size	
S1.BCNT		S1.RND	
Bytecount at completion		Data file random adr	
S1.RND+2		S1.RAD	
Data file random adr		KEY/RECORD ADDRESS	
S1.KSZ		S1.IAD	
KEY/RECORD SIZE		INDEX STRING ADDRESS	
S1.ISZ			
INDEX STRING SIZE			

The function codes are (hex):

01H ISAM-READ "NEXT"
02H ISAM-WRITE
2BH ISAM-READ
2CH ISAM-READ "PREVIOUS"
2DH ISAM-READ "LAST"
2EH ISAM-READ "FIRST"
2FH ISAM-DELETE
30H ISAM-UPDATE

Format is always image binary (see section 8).

The Return status codes are (decimal):

Code	Meaning
120	key not found
121	duplicate keys
122	illegal key
123	mismatch at check read
124	index not found
125	illegal data RECORD LENGTH
126	end of memory in ISAM task
127	incompatible ISAM file version

In addition any operating system errors that occur during processing will produce a return status code.

ISAM READ: This function uses the user defined key (S1.KAD/S2.KSZ) and index (S1.IAD/S1.ISZ) to do a search in the ISAM file. If no index is specified (S1.ISZ=0), the index last accessed is used. If it is the very first time (no access done) then the first index of the file is used. If no key is specified (S1.KSZ=0), the first key of the index is used. ON a successful read operation, the random address field points out the data record in the data file (used on delete and update), and the data record is read into the user buffer.

ISAM READ NEXT/PREVIOUS/FIRST/LAST: The next/previous/first/last record (by key) is read into the user buffer. If no index is specified, the index last accessed is used.

ISAM WRITE: The record specified by the user (S1.BAD/S1.BSZ) is written to the data file. All indices in the ISAM file are updated.

ISAM DELETE: The record specified by the user (S1.BAD/S1.BSZ) is compared to the record last read by the user. The records must be equal, or an error will result. The keys referring to that record are removed from the ISAM file.

ISAM UPDATE: The record specified by the user (S1.BAD/S1.BSZ) is compared to the record last read by the user. The records must be equal, or an error will result. The record specified by (S1.KAD/S1.KSZ) then replaces the old record in the data file. Each index is updated (if necessary) to reflect the new data record.

APPENDIX A

SYSTEM MNEMONICS AND ABBREVIATIONS

APPENDIX A
SYSTEM MNEMONICS AND ABBREVIATIONS

Item	Meaning
CCB	Channel Control Block
CDT	Channel Description Table
DCB	Device Control Block
DDT	Device Description Table
DMT	Device Mnemonic Table
DRT	Device Reference Table
EDT	Extended Descriptor Table
FCB	File Control Block
FMT	File Mnemonic Table
FRT	File Reference Table
ICB	Interrupt Control Block
IDT	Interrupt Description Table
RCB	Resource Control Block
RDT	Resource Descriptor Table
RMT	Resource Mnemonic Table
RRT	Resource Reference Table
SPT	System Pointer Table
STB	System Table
SVC	Supervisor Call
SYP	System Pointer Table Structure
TCB	Task Control Block
TDT	Task Description Table
TMT	Task Mnemonic Table
TRT	Task Reference Table
VCB	Volume Control Block
VDT	Volume Description Table
VMT	Volume Mnemonic Table
VRT	Volume Reference Table

APPENDIX B

ERROR CODES

APPENDIX B
ERROR CODES

COMMON ERRORS

OCT	HEX	DEC	SYMBOLIC	ERROR TEXT
000	00	0	SOS.OK	No error.
001	01	1	SOS.EON	End of nodes.
002	02	2	SOS.IFC	Invalid function code.
003	03	3	SOS.PRO	Can't connect at unconditional proceed.
004	04	4	SOS.OFFL	Off line.
005	05	5	SOS.PRES	Not present in this system.
006	06	6	SOS.NYET	Not yet implemented function.
007	07	7	SOS.CAN	Request is cancelled.
010	08	8	SOS.SVC	Invalid SVC function.

SVC-1 I/O ERROR CODES

OCT	HEX	DEC	SYMBOLIC	ERROR TEXT
012	0A	10	S1S.LU	Illegal LU, LU not assigned.
013	0B	11	S1S.AM	Invalid access modes.
014	0C	12	S1S.TOUT	Time-out.
015	0D	13	S1S.DWN	Device down.
016	0E	14	S1S.EOF	End-of-file.
017	0F	15	S1S.EOM	End-of-media.
020	10	16	S1S.RER	Recoverable error.
021	11	17	S1S.UNR	Unrecoverable error.
022	12	18	S1S.RND	Invalid random address.
023	13	19	S1S.NRND	Non-existent random address.

SVC-2 SUBFUNCTION ERRORS

OCT	HEX	DEC	SYMBOLIC	ERROR TEXT
024	14	20	S2S.ISB	Illegal subfunction number.

APPENDIX B - ERROR CODES

SVC-2.1 MEMORY HANDLING ERRORS

OCT	HEX	DEC	SYMBOLIC	ERROR TEXT
025	15	21	S2S.1PAR	Illegal parameter.
026	16	22	S2S.1EOM	End of memory.

SVC-2.3 PACK FILE DESCRIPTOR ERRORS

OCT	HEX	DEC	SYMBOLIC	ERROR TEXT
025	15	21	S2S.3IFD	Invalid file descriptor, syntax error.

SVC-2.4 PACK NUMERIC DATA ERRORS

OCT	HEX	DEC	SYMBOLIC	ERROR TEXT
025	15	21	S2S.4OFL	Overflow.
026	16	22	S2S.4NCV	Nothing converted.

SVC-2.7 FETCH/SET DATE/TIME ERRORS

OCT	HEX	DEC	SYMBOLIC	ERROR TEXT
025	15	21	S2S.7DAT	Invalid date.
026	16	22	S2S.7TIM	Invalid time.

SVC-2.8 SCAN MNEMONIC TABLE ERRORS

OCT	HEX	DEC	SYMBOLIC	ERROR TEXT
025	15	21	S2S.8CMD	Undefined command mnemonic.

SVC-2.12 OPEN/CLOSE DEVICE ERRORS

OCT	HEX	DEC	SYMBOLIC	ERROR TEXT
025	15	21	S2S.12AS	Device is assigned, can't be closed.
026	16	22	S2S.12DE	Device not found.
027	17	23	S2S.12IS	New volume already present.
030	18	24	S2S.12ON	Directory device not in close state.

SVC-3 TIMER ERRORS

OCT	HEX	DEC	SYMBOLIC	ERROR TEXT
036	1E	30	S3S.PAR	Invalid timer parameter.

SVC-4 TASK DEVICE ERRORS

OCT	HEX	DEC	SYMBOLIC	ERROR TEXT
050	28	40	S4S.ASGN	Not assigned.
051	29	41	S4S.TYPE	Invalid device type.

SVC-5 LOADER ERRORS

OCT	HEX	DEC	SYMBOLIC	ERROR TEXT
062	32	50	S5S.TID	Illegal task-id.
063	33	51	S5S.PRES	Task present.
064	34	52	S5S.PRIO	Illegal priority.
065	35	53	S5S.OPT	Illegal option.
066	36	54	S5S.CODE	Illegal code/item at load.
067	37	55	S5S.SIZE	Overlay doesn't fit.

SVC-6 TASK ERRORS

OCT	HEX	DEC	SYMBOLIC	ERROR TEXT
074	3C	60	S6S.TID	Illegal task-id.
075	3D	61	S6S.PRES	Task present.
076	3E	62	S6S.PRIO	Illegal priority.
077	3F	63	S6S.OPT	Illegal option.
100	40	64	S6S.EQUE	Event queue disabled.
101	41	65	S6S.STAT	Invalid task status.
102	42	66	S6S.QPAR	Invalid termination parameter.
103	43	67	S6S.QITM	More items present in event queue.
104	44	68	S6S.TYPE	Invalid task type.

SVC-7 FILE ERRORS

OCT	HEX	DEC	SYMBOLIC	ERROR TEXT
106	46	70	S7S.ASGN	Assignment error, double assign.
107	47	71	S7S.AM	Illegal access modes.
110	48	72	S7S.SIZE	Size error.
111	49	73	S7S.TYPE	Type error.
112	4A	74	S7S.FD	Illegal file descriptor.
113	4B	75	S7S.NAME	Name error.
114	4C	76	S7S.KEY	Invalid key.
115	4D	77	S7S.FEX	File exist error.

SVC-8 RESOURCE ERRORS

OCT	HEX	DEC	SYMBOLIC	ERROR TEXT
120	50	80	S8S.ID	Illegal resource-id.
121	51	81	S8S.CLAS	Invalid resource class.
122	52	82	S8S.PRES	Resource already present.
123	53	83	S8S.PRNT	Parent not present.
124	54	84	S8S.DUAL	Dual DCB not present.
125	55	85	S8S.RCB	Invalid RCB-type.
126	56	86	S8S.EOM	End-of-memory.

APPENDIX C

SVC FUNCTIONS AND BIT PATTERNS

APPENDIX C
SVC FUNCTIONS AND BIT PATTERNS

SVC1: MEMORY HANDLER

Function	Bit-Pattern	Use
S0F.WAIT	..0...xx	Read-write bits.
S1F.READ	..0...00	Wait for completion.
S1F.WRIT	..0...01	Read request.
S1F.WRD	..0...10	Write request.
	..0...11	Write with read check (device dependent).
	..0.xx..	Format bits. These bits indicate the type of data formatting requested.
S1F.IASC	..0.00..	Image ASCII.
S1F.FASC	..0.01..	Format ASCII.
S1F.IBIN	..0.10..	Image binary.
S1F.SPEC	..0.11..	Special.
	..0x....	Sequential-random bit.
	..00....	Sequential. The next logical record is to be accessed.
S1F.RND	..01....	Random. The logical record specified by the random address field is to be accessed.

Function	Bit-Pattern	Use
	..1xxxxx	Command codes. Codes not specified are resource dependent.
S0F.TST	..100000	Test request.
S0F.CAN	..100001	Cancel request.
S1F.FR	..100010	Forward record.
S1F.FF	..100011	Forward file.
S1F.WF	..100100	Write filemark.
S1F.BR	..100101	Back record.
S1F.BF	..100110	Back file.
S1F.RW	..100111	Rewind.
S1F.ATTN	..101000	Attention.
S1F.FEOF	..101001	Fetch end-of-file position.
S0F.NW	.x.....	Wait-proceed bit.
S0F.PRO	x.....	Unconditional proceed bit.

SVC2.1: INPUT/OUTPUT

Function	Bit-Pattern	Use
S2F.1ALO	..000001	Allocate memory.
S2F.1MAX	..000010	Reserved.
S2F.1REL	..000011	Release memory.
S2F.1TCB	..000100	Allocate a TCB, only internal use.
S2F.1CAN	..000101	Remove callers TCB, only internal use.

SVC2.2: LOG MESSAGE

Refer to the SVC1 Function Code parameters regarding data formatting.

SVC2.3: PACK FILE DESCRIPTOR

Function	Bit-Pattern	Use
S2F.3FN	..00...1	Unpack as filename, if not specified.
S2F.3KEP	..00..1.	Keep non-modified fields.
S2F.3CNT	..00.1..	String size specified.
S2F.3PMO	..001...	Pack modifier.

Termination Function	Bit-Pattern	Meaning
S2T.3NEL	0000...1	Element name not found.
S2T.3NFN	0000..1.	File name not found.
S2T.3NVO	0000.1..	Volume name not found.
S2T.3NMO	00001...	Modifier not found (only set if S2F.3PMO is requested, and no modifier found).

SVC2.4: PACK NUMERIC DATA

Function	Bit-Pattern	Use
	..00..xx	Conversion base.
S2F.4DEC	..00..00	Decimal.
S2F.40CT	..00..01	Octal.
S2F.4HEX	..00..10	Hexadecimal.
	..00.x..	Sign handling.
	..00.0..	No signed input allowed.
	..00.1..	Input may be signed.
S2F.4SGN		
	..00x...	Destination.
	..000...	In parameter block.
	..001...	Address specified.
S2F.4IND		

SVC2.5: UNPACK NUMERIC DATA

Function	Bit-Pattern	Use
	xx	Converting base.
S2F.5DEC	00	Decimal.
S2F.5OCT	01	Octal.
S2F.5HEX	10	Hexadecimal.
	.....x..	Sign handling.
	.....0..	Unsigned conversion.
S2F.5SGN	.....1..	Signed conversion.
	....x...	Source description.
	....0...	Number in parameter block.
S2F.5IND	....1...	Address specified.
	...x....	Space flag.
	...0....	Leading zeros.
S2F.5SP	...1....	Leading spaces.
	..x.....	Field justifying.
	..0.....	Right justify.
S2F.5LFT	..1.....	Left justify.

Size	Function	Meaning
S2Z.5ASC	xxxx	Number of bytes in ASCII-string.
S2Z.5BIN	.xxx....	Number of bytes in binary number.
S2Z.5INC	1.......	Auto-increment of binary pointer.

SVC2.7: FETCH OR SET DATE AND TIME

Function	Bit-Pattern	Use
S2F.7GET	..0...01	Fetch function.
S2F.7SET	..0...10	Set function.
S2F.7SLC	..0.00..	Slice handling.
S2F.7DAT	..0..1..	Date handling.
S2F.7TIM	..0.1...	Time handling.
	..00....	ASCII data, not at slice handling.
S2F.7BIN	..01....	Binary data, not at date and time.

SVC2.8: SCAN MNEMONIC TABLE

None.

SVC2.12: OPEN/CLOSE DEVICE

Function	Bit-Pattern	Use
S2F.12CL	..000001	Close.
S2F.12OP	10	Open.
S2F.12PR	.....1..	Write protected.
S2F.12NF	....1...	Non-file structured.
S2F.12AD	...1....	SVC-handler address specified, only directory oriented devices.
S2F.12AL	..1.....	Fetch auto start line. Only vaild at open file structured.

SVC.3: TIMER COORDINATION

Function	Bit-Pattern	Use
	..0000xx	Time specification:
S3F.MIL	..000001	Milliseconds.
S3F.SEC	..000010	Seconds.
S3F.TOD	..000011	Time of day.
	..100xxx	Commands:
SOF.TST	..100000	Test request.
SOF.CAN	..100001	Reserved.
S3F.CMIL	..000010	- " -.
S3F.CSEC	..000011	- " -.
S3F.CTOD	..100100	- " -.
	.x.....	Wait-proceed bit.

SVC 4: TASK DEVICE HANDLING

Function	Bit-Pattern	Use
S4F.TRIG	..0000.1	Trigger initiator.
S4F.CAN	..00001.	Set cancel pending.

SVC 5: OVERLAY LOADER

Function	Bit-Pattern	Use
S5F.LOAD	..00...1	Load.
S5F.STRT	..00..1.	Start overlay.
S5F.ABS	..00.1..	Absolute start address at overlay start.
S5F.OVL	..001...	Overlay handling, else task handling.
SOF.NEW	.x.....	Wait-proceed bit.
SOF.PRO	x.....	Unconditional proceed bit.

SVC 6: TASK CONTROL

Function	Bit-Pattern	Use
S6F.LOAD	..000..1	Load task.
S6F.STRT	..000.1.	Start task.
S6F.ABS	..0001..	Absolute start address.
S6F.QTST	..001000	Test event queue.
S6F.QWAI	..0010.1	Wait for queue event.
S6F.QTRM	..00101.	Terminate event.
S6F.QDIS	..001100	Disable event queue.
S6F.QENI	..001101	Enable event queue.
S6F.SUSP	..001110	Suspend myself.

...Continued...

SVC 6: TASK CONTROL (Cont.)

Function	Bit-Pattern	Use
S0F.TST	..100000	Test task.
S0F.CAN	..100001	Cancel task.
S6F.PAUS	..100010	Pause task.
S6F.CONT	..100011	Continue task.
S6F.PRIO	..1001.1	New task priority.
S6F.OPT	..10011.	New task option.
S6F.TSKW	..101000	Wait for task termination.
S6F.ADDQ	..101001	Add to event queue.
S6F.STSW	..101010	Wait for task status change.
S6F.TYPE	..101011	New task type.
S0F.NW	.x.....	Wait-proceed bit.
S0F.PRO	x.....	Unconditional proceed bit.

Option	Bit-Pattern	Meaning
S60.DASG	0000...1	Default assign allowed.
S60.NSTK	0000..1.	No stack check.

The option field is also used at function S6F.TYPE in which case:

Option	Bit-Pattern	Meaning
S6T.RES	0000..1	Set the task memory resident.
S6T.NAB	0000.1.	Set the task non-abortable from other tasks.

SVC 7: FILE HANDLING

Function	Bit-Pattern	Use
S7F.ALLO	..0....1	Allocate.
	..0...1.	Reserved.
S7F.ASGN	..0..1..	Assign.
S7F.DELC	..0.1...	Delete at close.
S7F.CLOS	..01....	Close.
S0F.TST	..100000	Test request.
S0F.CAN	..100001	Cancel all previous requests.
	..100010	Reserved.
S7F.CHKP	..100011	Checkpoint.
	..100100	Reserved.
	..100101	- " -.
S7F.RNAM	..100110	Rename.
S7F.FAT	..100111	Fetch attributes.
	.x......	Wait-proceed bit.
	x.......	Unconditional proceed bit.

S7.TAM, Access Mode

The low nibble of this byte specifies the Access Priviliges. If the access mode for a direct access file is 'SW', it will be changed to 'EW'!

Access	Bit-Pattern	Meaning
S7A.SRO	000	Sharable Read Only.
S7A.ERO	001	Exclusive Read Only.
S7A.SWO	010	Sharable Write Only.
S7A.EWO	011	Exclusive Write Only, will position to end-of-file.

SVC 8: RESOURCE HANDLING

Function	Bit-Pattern	Use
S8F.EST	..000.01	Establish resource.
S8F.RMOV	..000010	Remove resource, only available to the owner.
S8F.TEST	..000011	Test the presence of a resource.
S8F.NRCB	..0001..	RCB already present.

S8.CLAS, Class

This field contains the resource class

Class	Bit-Pattern	Meaning
S8C.DEV	001	Devices.
S8C.TSK	010	Tasks.
S8C.COM	011	Common.
S8C.VOL	100	Volumes.
S8C.SVC	101	SVC-functions.
S8C.SVC2	110	SVC2-subfunctions.

S8.TYPE, Type

This field contains the resource type.

Type	Bit-Pattern	Meaning
RTT.PURE	00	Shared resource.
RTT.RCB	01	Exclusive resource.
RTT.RRT	10	Dummy, S8.ADR points at a new resource.
RTT.AREA	11	Area, no entry.
RTT.DIR	.... .1..	Directory oriented.

Type	Bit-Pattern	Meaning
RTT.SVC	.... 1...	Entry at ALL SVC-calls.
RTT.OFFL	...1	Off line, not accessible.
RTT.PROT	..1.	Protected, write not allowed.
RTT.NFST	.1..	Non-file structured, only for system's use.
RTT.NRMV	1...	Resident, non-removable.

RDT.TYPE, TYpe

Specifies the false of resource.

Type	Bit-Pattern	Meaning
RCT.RCB	.... .000	No special type.
RCT.DCB	.... .001	DCB, Device Descriptor Table present.
RCT.TCB	.... .010	TCB, Task Descriptor Table present.
RCT.FCB	.... .011	Only for system's use.
RCT.VCB	.... .100	VCB, Volume Descriptor Table present.
RCT.AREA	.... .101	Area, no entry.
RCT.PRNT	.... 1...	Coordination parent specified.
RCT.DESC	...1	Ony for system's use.
RCT.NW	..1.	Don't support no-wait functions.
RCT.PRO	.1..	Don't support un-conditional proceed.
RCT.NAB	1...	Non-abortable, can't be cancelled.

TDT.TYPE, Task Type

Describes the type of task:

Type	Bit-Pattern	Meaning
TCT.RES	1	Set the task memory resident.
TCT.NAB	.... ..1.	Set the task non-abortable for other tasks.

TDT.OPT, Task Options

Hold the task options.

Option	Bit-Pattern	Meaning
TCO.DASG	1	Default assign allowed.
TCO.NSTK	.... ..1.	No stack check.
TCO.EMSG	.... .1..	Error message print-out by the system.
TCO.RCOV	.... 1...	System recovery.

DDT.ATTR, Attributes on the Device

This is a bit pattern which describes the device attributes.

Attribute	Bit-Pattern	Meaning
ATR.READ	1	Read.
ATR.WRIT	....1.	Write.
ATR.FASC	.... .1..	Formatted ASCII.
ATR.SPEC	.... 1...	Special formatting.
ATR.RND	....1	Random access.
ATR.IACT	....1.	Interactive device, echo of input.
	.... .1..	Reserved for future use.
	.... 1...	Reserved for future use.
	....1	Reserved for future use.
ATR.FR	.... ..1.	Forward record.
ATR.FF	.... .1..	Forward file.
ATR.WF	.... 1...	Write file-mark.
ATR.BR	...1	Back space record.
ATR.BF	..1.	Back space file.
ATR.RW	.1..	Rewind.
ATR.ATTN	1...	Attention.

DDT.TYPE, Device Type

Device Type	Bit-Pattern	Meaning
DCT.ICB	1	Interrupt Control Block (ICB) present.
DCT.DEDI	.... ..1.	Dedicated interrupt service.
DCT.ENI	.... .1..	Reserved.
	.... 1...	Reserved.
DCT.TASK	...1	Indicates a task device.
DCT.DUAL	..1.	dual DCB information.

GLOSSARY OF TERMS

GLOSSARY OF TERMS

ASCII Data	Seven bit data with the most significant bit in the byte cleared.
Binary Data	Eight-bit data.
Bit Management Subroutines	Subroutines which allocate and delete files on direct-access-volumes.
Byte Access	When data in a file is byte transferred irrespective of record length. Data transfer normally occurs in binary format. Any byte in the file can be accessed and I/O is performed in the same way as the logical transfer of files with variable record length.
Byte Random Access	Ability to retrieve any byte from any record in a file wihtout having to specify the record number.
Cache Memory Sector Buffering	A Disk Buffer Management method which reduces disk accesses.
Channel Descriptor Table (CDT)	Table used by the operating system to create a Channel Control Block (CCB) for a device. The CDT is a Continuation of the IDT.
Command	Basic unit of conversation between a terminal user and the operating system.
Connection Handler	Passes a Parameter Block request to the appropriate operating system handler.

Console Manager Task	A program which runs the console.
Contiguous File	A file which is stored on contiguous sectors on a disk.
Current Task	Any task that is currently executing instuctions.
Device Control Block (DCB)	Used by the operating system to specify the characteristics of each device.
Device Descriptor Table (DDT)	Used by the operating system to specify both real devices and task devices. The DDT is a continuation of the RDT.
Device Driver	A program which controls a physical device.
Direct Access Device	A disk.
Directory Management Subroutines	Subroutines which maintain information on all currently allocated files.
Disconnection Handler	Takes the Parameter Block from the system subroutine and returns it to the task which made the initial request.
Dispatch Priority	Priority set up by the Operating System which determines the order in which the task is to be serviced.
Dormant Task	Any task that has not been started.

Driver Initiator	A routine which is called whenever a request is sent to a driver. It must enable the various I/O parts and interrupts used by the driver.
Driver Interrupt	That portion of the Driver Code which responds to the interrupt as directed by the interrupt handler.
Exclusive Resource	Any resource that can be used by only one task at a given time.
Executive	The operating system.
Executor	A system subroutine.
Extended Descriptor Table (EDT)	Used by a programmer to expand a control block and initialize it with some data. The EDT is an extension of the RDT.
File Manager	A software package that supports the File Management System.
Fixed Record Length File	A file, all of whose records have the same length.
Hardware Driver	A program which controls a piece of hardware.
Idle Loop	A procedure which is executed when the Operating System has no other task to run.

Indexed-File	A file which is distributed over many non-contiguous sectors of a diskette. There is a pointer to each sector occupied by the file which indexes the file.
Initial Value Table (IVT)	Part of the Operating System Initializaiton Code which supplies configuation addresses.
Input/Output Management	The interaction of tasks with Input/ Output devices.
Interrupt	A control signal through which external logic can demand the attention of the operating system.
Interrupt Control Block (ICR)	Is used for the interrupt porition of the operating system.
Interrupt Descriptor Table (IDT)	Is used by the operating system to describe the interrupt side of a device. The IDT is a continuation of the DDT.
Interrupt Handler	A common piece of code that responds to all interrupts and decides which driver is appropriate to activate.
Logical Access	When data in a file is transferred as logical records. System data buffering is used.

Node	Part of an operating system structure which is used to hold various pieces of information which are function dependent. It can be included in prioritized lists.
Nonresident Task	Any task which is removed from memory after execution.
Operating System Mangement	The preparation, management, and execution of task files.
Parameter	An element in a parameter block.
Parameter Block	A block of code which contains the parameter of an SVC.
Paused Task	Any task that has been paused during execution.
Physical Access	When data in a file is transferred as physical records. No buffering or formatting is needed, because the data is transferred in the byte blocks without using the File Manager.
Program Status Word	Indicates the status of the program currently running on the operating system. This is not the PSW Flag Register of the CPU chip.

Ready Task	Any task that is ready to become the current task.
Relocatable Load Module	A piece of object code that can be relocated anywhere in memory (normally by the Relocatable Loader).
Resident Task	Any task which is not removed from memory after completing execution.
Resource	A process or an area of memory that can be used by a task.
Resource Control Block (RCB)	The portion of the operating system code which coordinates operating system resources.
Resource Descriptor Table (RDT)	Used by the SVC handler to create the control blocks necessary to handle exclusive resources.
Resource Mnemonic Table (RMT)	Used by the operatng system to give symbolic definitions of the resources that are to be used. The RMT contains pointers to the RRT.
Resource Reference Table (RRT)	Used by the operating system to give a numeric description of a resource. The RRT contains the number of the resource and the type of resource.

Run Mode	When the CPU is executing instructions.
Shared Resource	Any resource that can be used by several tasks at the same time.
Software Driver	A program which executes a subroutine.
Stop Mode	The actual halt state of the CPU chip. Can only be exited by an external interrupt.
Strict Priority Scheduling	When a task remains active until it gains control of the processor.
Subfunctions	Parameters in the SVC 2 Parameter Block that allow the user to perform certain tasks like communicating with the console operator, allocating memory, and so on.
Supervisor Calls (SVC's)	Assembly Language programs which are used be the Monroe Operating System to perform Operating System Service.
SVC-Functions	Those functions which are performed by the SVC's.
SVC-Handler	A common piece of code that responds to all SVC requests and decides which SVC to execute.
Symbiont	A task device. It operates as a normal task and uses its event queue to accept requests from other tasks.

System Pointer Table (SPT)	A set of pointers which index additional operating system structure located at arbitrary points in memory.
System Queue Handler	To be supplied.
Task	The funcamental unit of work in the Monroe Operating System. It can consist of a single program or of a main program together with a number of subroutines and overlays.
Task Control Block (TCB)	Any task within the system (active or dormant) requires at least one task Control Block. The TCB give information about the task such as stack size, starting address, open files, etc.
Task Descriptor Table (TDT)	Used by the operating system to create a Task Control Block (TCB) which is then used to Control a task. The TDT is a Continuation of the RDT.
Task Priority	The priority currently assigned to a task.
Termination Handler	Used within a interrupt routine. Is called when the routine is completed.
Time Slice Scheduling	When task of equal priority receive shares of CPU time.

User-File	An element or file in the User-File Directory.
User-File-Directory	A subdirectory of the Master-File-Directory.
User Mode	The mode in which all user tasks run.
Variable Record Length File	Any file consisting of records whose lengths vary.
Volume	A directory oriented mass storage device.
Waiting Task	Any task that is waiting for an event.

INDEX

READER COMMENT FORM DATE _____

Your comments and suggestions help to improve this publication.
Please complete the questionaire. Fold, staple, and mail it to Monroe.

Name_____ Title_____
Organization_____
Street_____ State_____ Zip_____

Publication Title_____
Publication No._____ Revision Letter_____ Date_____

CIRCLE YOUR RESPONSES TO THE STATEMENTS BELOW. IF YOU RESPOND "NO" TO A STATEMENT, ENTER THE STATEMENT NUMBER AND THE PAGE AND PARAGRAPH IN THE PUBLICATION THAT PROMPTED YOUR RESPONSE.

1. The publication was used for
   Learning Installing
   Reference Maintaining
   Sales Programming

2. The user/reader was
   High-level Programmer
   Occasional Programmer
   Student Programmer
   Data Entry Operator
   Other (specify)_____

3. The material is accurate. YES NO
4. The material is clear. YES NO
5. The material is complete. YES NO
6. The material is well organized. YES NO

ENTER DETAILED INFORMATION FOR STATEMENTS 3-6.

Statement No.	Page No.	Paragraph No.	Comments

7. The overall rating for this publication is
   Very Good Good Fair Poor Very Poor

   Briefly explain your rating._____

8. Additional comments_____

FOLD

Software Publications Dept.

No Postage
Necessary
if Mailed in the
United States

BUSINESS REPLY MAIL

FIRST CLASS PERMIT NO. 731 MORRISTOWN, N. J.

POSTAGE WILL BE PAID BY ADDRESSEE

MONROE division of
Litton Business Systems, Inc.
Box 9000R
Morristown, N.J. 07960

CUT ALONG LINE

FOLD

STAPLE

STAPLE